A critical appraisal of data base management

Potts, I.

A CRITICAL APPRAISAL

OF

DATA BASE MANAGEMENT

I. POTTS

A B S T R A C T

The original purpose of the thesis was to compare the popular data base management systems in existence:- IMS, TOTAL, SYSTEM 2000, IDMS, ADABAS and DMS 1100.

At the commencement of the exercise it became clear that a set of data base management characteristics and user requirements were needed as a basis for the comparison, and indeed, the documenting of the characteristics forms a major part of the exercise, bearing in mind the necessity to become familiar with the concepts and with the views of the experts in the subject. The bibliography presents the basis for this education process.

In the course of this learning process, it became clear that data base management systems are only part of the data management 'picture' in any data processing environment. The way that data processing management regard the use of such software; the importance of influencing user management and personnel; the way the structure of data processing staffing may change and the whole evolutionary process that commences when data is regarded as an organisation's resource rather than the object of processing.

With the expected impact of future technological change in computing hardware and the numerous and varied solutions that become available to data processing management, it will be necessary for future data base systems to provide flexibility in data handling.

It is the main purpose of this thesis to set out the requirements of the management of data, in its structuring, organisation, access and support, and to describe how the DATA MANAGEMENT environment of the 1980's and 1990's may evolve from the DATA PROCESSING environment of the 1960's and 1970's.

In this context, the comparing of present day software packages which are first steps along the path to this new environment becomes less relevant.

A CRITICAL APPRAISAL

OF

DATA BASE MANAGEMENT

I. POTTS

PREFACE

The thesis is split into 2 parts. The first part describes the characteristics of data base management. The second part describes a possible data base management system that would be suited to the future technological developments of the next 10 to 20 years.

Chapter 1 provides an introduction to data base management by detailing the historic developments of data storage and capture, stressing in general terms the characteristics of data management.

Chapter 2 describes data, the way it is required to be structured and organised for access to the data and for data storage.

Chapter 3 consists of a detailed description of the data base management functions and characteristics of data handling.

Chapter 4 identifies two types of system, the normal operation systems that are used in every-day commercial data processing, and information systems which will become very important in the future but which in past have been implemented only rarely and relatively crudely.

Chapter 5 describes the organisational and user environment as it is affected by the use of data base management systems.

Chapter 6 describes in brief detail three different theoretical proposals for the management and handling of data. This chapter ends the first part.

Chapter 7 as the first chapter in the second part, identifies the way ahead for data processing (or management service) departments and how best should such departments approach the implementation of their first data base projects.

Chapter 8 describes the first part of the conceptual data base management system, the syntax of the data language required to describe each of the levels in the system. Included in the chapter is the description of an application used throughout the remainder of the thesis as a means of illustrating the system.

Chapter 9 defines how the data base management system would be implemented.

Chapter 10, in conclusion, stresses the significance of the system applied to future technological developments.

There are three appendices. The first describes the Bachman Diagram method of describing structures, culminating in the definition of the 3-level data description described in the thesis. The second appendix defines the implementation of the temporary file facility used frequently within the thesis. The final appendix defines, in notational form, the data language syntax for the conceptual data base management system described in chapter 8.

A bibliography lists over 50 articles, books and reports which were read by the author during the study.

Certain points are worth noting before reading any further. The phrases 'data base management system' and 'data base administrator' are used so often that 'D.B.M.S.' and 'D.B.A.' respectively are used in general.

Any reference to the male gender (e.g. 'his', 'him') is equally applicable to the female gender.

The use of the word 'data' (plural of 'datum') is used in the singular context throughout the thesis.

Finally I wish to express thanks to those who have given invaluable assistance throughout.

Mr. Roper, my study supervisor, for his guidance and advice.

Mr. J. Golightly whose assistance and support, when applying for the course and throughout the major part of the study, is greatly appreciated.

Most of all, I am indebted to my thesis typist Mrs. Hope who patiently typed and corrected this thesis and to many others who gave help, thank you.

C O N T E N T S

PART ONE

1. INTRODUCTION

1.1. 100 YEARS OF DATA USAGE.

In the middle of the last century data processing was done by a clerk who maintained the well bound ledgers dealing with every company transaction. A simple process which was easily understood. All company information was at the clerk's finger tips even to the point of balancing the books at the end of the day.

As the small company grew, there evolved too much clerical work for one clerk, there became the need for several clerks each dealing with a particular section of the book keeping, for instance, one clerk specialised in the sales ledger.

During this century, as certain parts of the data processing function became mechanised, transactions were punched on cards and batched. Magnetic tape permitted transaction batching on a much greater scale. The batching was a concession to the way the data processing machine worked, the way in which a complete process was performed on all the data e.g. a file update. The data was held on separate files. What data was held and the way it was held was defined by the way the data process worked and what data that process required.

The data base approach is now returning us to the method of working of that clerk 100 years ago, with the benefits of the data being up-to-date and queries quickly answered for large 20th. century organisations.

1.2. A HISTORY OF ELECTRONIC DATA PROCESSING.

With the introduction of the computer, the way data was handled by an organisation introduced several extra steps to enable data to be handled by this new machine. The benefits of speed of handling, speeds beyond the imagination only a few years before, forced large organisation in particular to fit their own methods of handling the organisation's information to the way the data, on which this information was based, was being quickly and automatically processed.

How did the way the data was processed change to accommodate the way an organisation needs to control and use its information?

At the start of this electronic revolution, all data was input to the computer by means of batches of cards with the data represented on the cards as the presence or absence of a hole in a certain position on the card. The information had effectively been converted by a number of human and mechanical processes to the computer's 2-state system. An enormous concession for the benefits of speed. Since that time a great deal of effort has been made to push the interface between man and machine as near to the level of man-to-man communication as possible. Printers provided information in readable form at an early stage in this development process, but for many years the requirement to convert input data to a 2-state system remained with the data processing industry. Efforts were first made to reduce the size of the problem. In particular, the bulky 80-column card, of which probably only 25% of the card was used, was often dropped on the floor and put out of sequence. Development of magnetic tape enabled the 'base' of data to be maintained on a machine readable medium that was much more compact and easy to handle, this meant that cards were used only for inputting transactions to the system. Electric punching machines brought the man/machine interface to the level of typing. These machines are still used to a great extent, providing extra facilities, for instance, interpretation and automatic punching. Of course this did not remove the card handling requirement, and so products were produced to batch transactions directly on to magnetic media from typewriting consoles. This was fine for a batch processing operation but soon there was a requirement to input the transaction as soon as the transaction became known and to input that transaction from the location of its creation rather than informing the data processing department directly. Such a facility is provided through a terminal with a keyboard similar to a typewriter. This type of operation is called transaction processing. Terminals are now provided to permit input of graphic data and voice communication.

Not only have methods of input changed, but also the way data is stored. Initially, all the information concerning the stored data was implicit within the program that processed the data. It was realised that for the enquiry type of program there are only a limited number of processing options, hence it would be possible to provide a generalised enquiry program with the particular enquiry option supplied by parametric input, and in this way, development times could be reduced. Gradually the relationships between data were built into the stored base of data. The best example of such data

storage is the 'bill of materials' file where the structure of data relationships represent the way the parts of a product relate to one another.

It is true to say that the way we handle data has hardly advanced from this point. The idea of data base management has been on the data processing horizon for most of this decade but it has seemed a dormant concept with only a limited number of restricted implementations. The main reason would seem to be the lack of the necessary technological development in hardware and software.

1.3. FUTURE DEVELOPMENTS IN STORAGE.

The implications of the new developments and technologies are dealt with in later chapters but it seems fitting to mention general implications in the developments in storage.

The growth rate in the size of computer storage is greater than the growth in size or power of any other data processing component. To match this growth, the cost of data storage hardware is dropping more rapidly than other costs in data processing, soon it will become cheaper to store data on computer files than to store them on paper. The industry is improving its capability to store other communication forms such as drawings, photographs and human speech. There is this see-saw effect of the capacity of storage going up, while the cost per bit of that storage continues to come down.

The spectrum of storage devices is likely to increase in the future. Currently most computer installations have 2 levels of on-line storage this will increase to 4 levels in a few years' time with the introduction of faster but smaller solid-state storage, and the development of electro-mechanical devices with access times of several seconds but with a capacity of the order of magnetic tape libraries.

1.4. THE CONCEPTS OF DATA BASE MANAGEMENT.

Prior to the data base concept, the centre pivot of all D.P. operation was the program which processed the data. The files used contained the data required to be processed by the program. Basically the data base concept looks at the situation in reverse with the data at the centre of the operation and the program acting on the data base.

In this sense the conventional approach of having several units each with a process (program) at the unit's centre is replaced by a single unit which is the base of data. In the conventional approach the data is subordinate to the program, but in the data base approach the programs are subordinate to the data (Fig. 1.1.).

With the data as one unit rather than several units fragmented between programs, the possibility of relating all data across an organisation can be realised by permitting access to that data across departmental boundaries. This DATA INTEGRATION can provide a number of advantages to the organisation because it provides a data resource which can be as important as any other organisation resource. In this way the integrated data can provide an information base which would be possible to only a limited extent if the data was fragmented, i.e. the maximum amount of information is related to the data accessed by any single program.

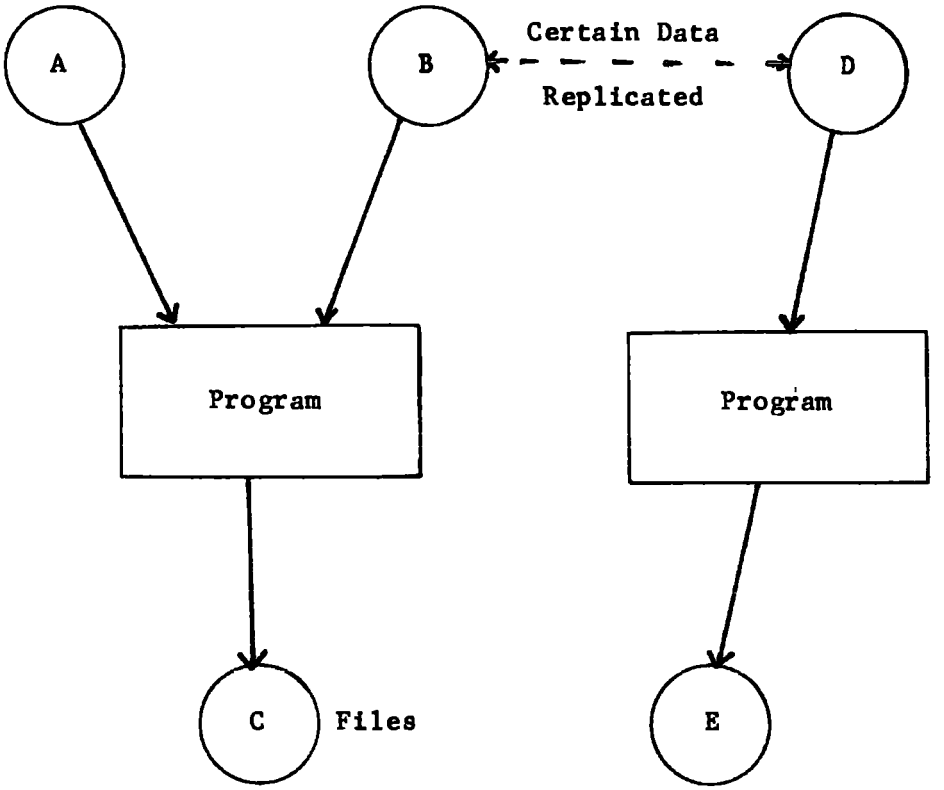
Such an information base can be accessed by generalised enquiry programs, similar to those described in Section 1.2, as well as the specifically written batch and transaction programs.

In the conventional approach, different programs' data requirements can overlap, so that data is stored more than once. Apart from the obvious extra cost in computer storage caused by this duplication of data, there is a major overhead in having to update all the copies of that data, but much more of a problem is created because all such copies of the data must be consistent at any point in time. Clearly this problem can be overcome by having to store data only once.

The data base should be independent of the programs that access this base. This means that if data relationships are changed within the data base, only those programs that know of the data relationships that have been changed are affected. Similarly the data base should be independent of the storage on which it is held thus permitting a development of hardware which does not affect the data and its relationships.

By having only one copy of the data a greater degree of control of the resource is possible, to prevent unauthorised access, to permit recovery from data loss in the case of hardware or software failure (particularly in on-line systems) and the ability to provide information about the information in the data base to document the data for those who wish to use the data.

THE CONVENTIONAL APPROACH



THE DATABASE APPROACH

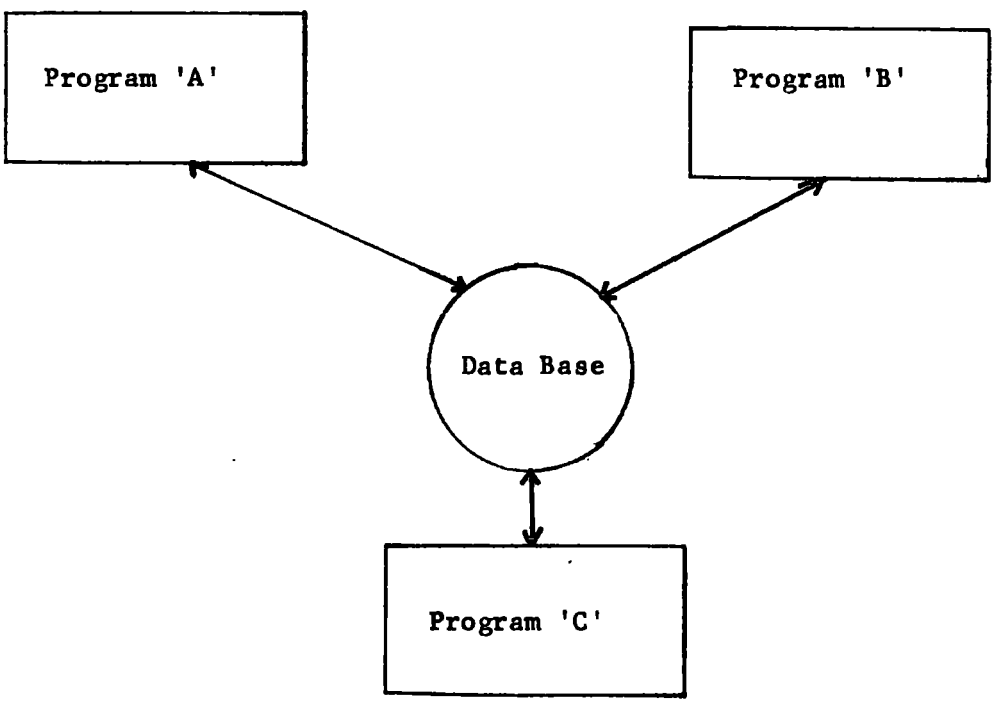


FIG. 1.1.

2. DATA.

2.1. INTRODUCTION.

The organisation of data and how best to access the data is central to the success of any data base environment or indeed of any on-line system requiring a certain response time.

The types of data organisation and data access methods are described in this chapter, but before looking at these aspects of data and its storage what are the properties of data?

2.2. THE NATURE OF DATA.

2.2.1. The Properties of Data.

There are five such properties of data:-

(1) Semantics.

The interpretation of meaning of data. There are 2 such meanings, the EXTENT of the meaning which is the specific definition of the item of data e.g. the dictionary definition of a word, and the INTENT of the meaning or sometimes referred to as the connotation. As an example of the differences in meanings, between extent and intent of a data item, a bank balance data item is called BAL, the extent of meaning is this name coupled with the symbol which represents the value associated with the data item.

Thus BAL = BV
Where BV = £1000

The extent of the meaning is "the bank balance, BAL has associated with it the symbol BV which has associated with it a value of £1000".

The intent is the interpretation that relates to the specific usage in the application, in this case "an amount of money BV, owed by an individual, who is represented in the system by a record".

(2) Syntax.

The association or relationship with other data. This is rather equivalent to the definition which defines how atoms combine to form molecules.

(3) Representation.

Once again this property has an explicit and an implicit form. An example of the explicit representation is BAL/£1000. An example of the implicit representation of data is the case of a customer balance. If the balance values are not recorded in the customer's record but rather the balance when required is calculated from the stored customer account transactions, the balance is represented implicitly as the sum of a set of quantities and its implicit representation is made explicit by performing certain processing.

(4) Hierarchy of Data Elements.

Data is built into increasingly more complex forms of data.

(5) Function.

This property differentiates between whether data is user derived and maintained or whether maintained and derived by the data management system.

Summarising the five properties:-

Semantics - the meaning;
Syntax - relationship to other data;
Representation - to enable storage, retrieval, processing or display of data;
Hierarchy - the organisation into larger structures;
Function - the intended use.

2.2.2. Data Items.

The smallest unit of data that expresses any sought of meaning to its users is called a DATA ITEM. To use a chemistry analogy, data items are the molecules of a data base with the bits and bytes the 'atoms' that go to make up the data items.

If different data items are grouped into a named grouping this group is called a DATA AGGREGATE. The grouping of more than one occurrence of the same data item is referred to as a VECTOR or ARRAY.

If data items and data aggregates are required to be processed together by a computer program they are grouped together into a logical processing unit, the RECORD. A program usually reads or writes whole records.

The data items and data aggregates have in the past been known as fields and one of these fields is used to identify the other fields which make up the record grouping, this identification field is called the KEY field. The records which contain the same types of data are themselves grouped into a FILE so that all customer records are grouped into a customer file and each record within the file is identified by a key data item or key data aggregate value. The term DATA ELEMENT is a general term which may be used for data items or aggregates.

2.2.3. Data Relationships.

One of the characteristics of data is how do data elements relate to other data elements? There are different varieties of relationships as follows:-

Comparison.

How a quantity compares with a re-ordering level quantity.

Context.

A street name can have a differing significance in differing circumstances but when part of an address it provides specific meaning.

Access.

Certain data can only be obtained by first knowing the value of other data e.g. the employee name can only be obtained by obtaining an employee number.

Definition.

When a data element is used to describe the height of a person, and if that element takes the value '6' a particular unit of measurement must be associated with the data value to indicate whether the height of the man is six inches, feet, yards or miles.

Association.

Pupils in a school belong to a particular class there is an association between a pupil and his class.

Ownership.

A component as part of a sub-assembly is 'owned' by that sub-assembly.

Data Relationships can be classified by their properties:-
Symmetric or asymmetric.

In a symmetric relationship a 2-way linkage between two data elements is meaningful. For example, a supplier address can be accessed given the supplier number, whereas the supplier number cannot be accessed from the supplier address, thereby being asymmetric.

Directed or Non-directed.

A relationship is directed if two data elements cannot be reversed without altering the relationship between them. The relationship between teacher and pupil is directional i.e. a HIERARCHY, whereas the relationship between stock item quantity-in-hand and the stock item re-order level is non-directed.

Transitive or Intransitive.

In Fig. 2.1, Diagram A shows a transitive relationship because there is meaning in a relationship between A and C in addition to the relationship between B and C.

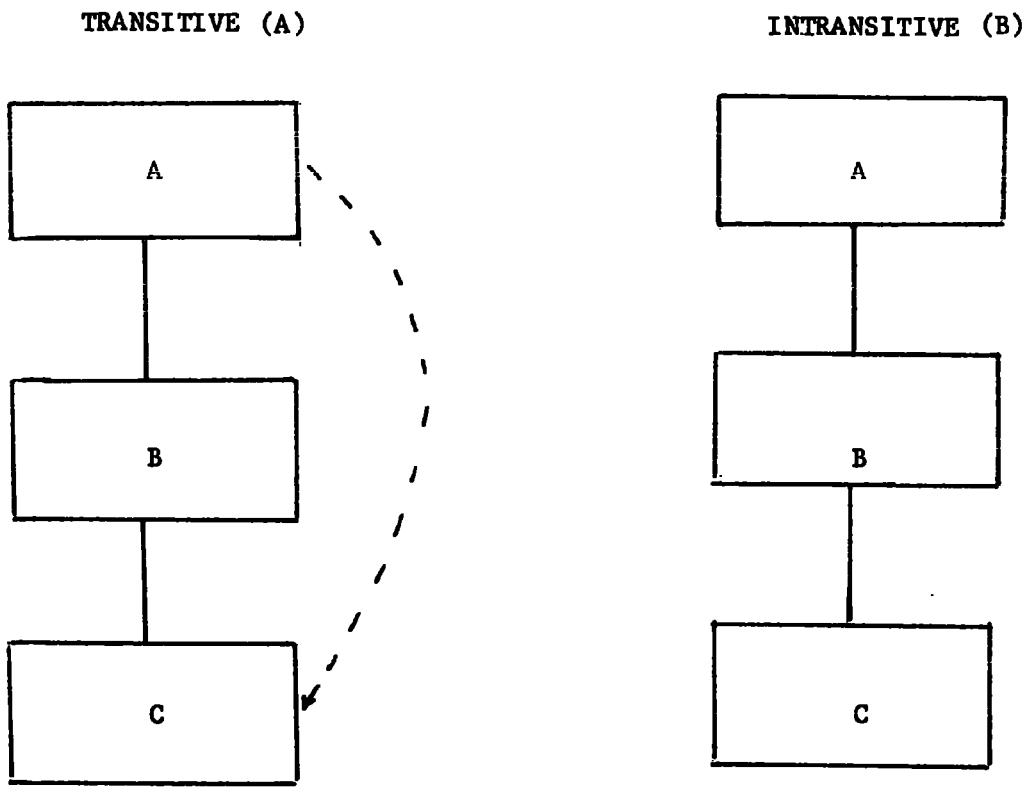


FIG. 2.1 - TRANSITIVE AND INTRANSITIVE DATA RELATIONSHIPS.

In this diagram a rectangular box is being used to represent a data element and the unbroken line between them indicates there is a relationship between the two data elements.

There is more information that can be provided about a data relationship. How many data elements of A relate to how many data elements of B? This characteristic is called the EXTENT of the relationship.

Simple - Simple Relationship.

Where one data element A relates to one data element B e.g. the relationship employee number and employee name is such a relationship, better known as a ONE-TO-ONE RELATIONSHIP. The extent is symbolically described by arrows on the unbroken line. Fig. 2.2. describes this one-to-one relationship between A and B.

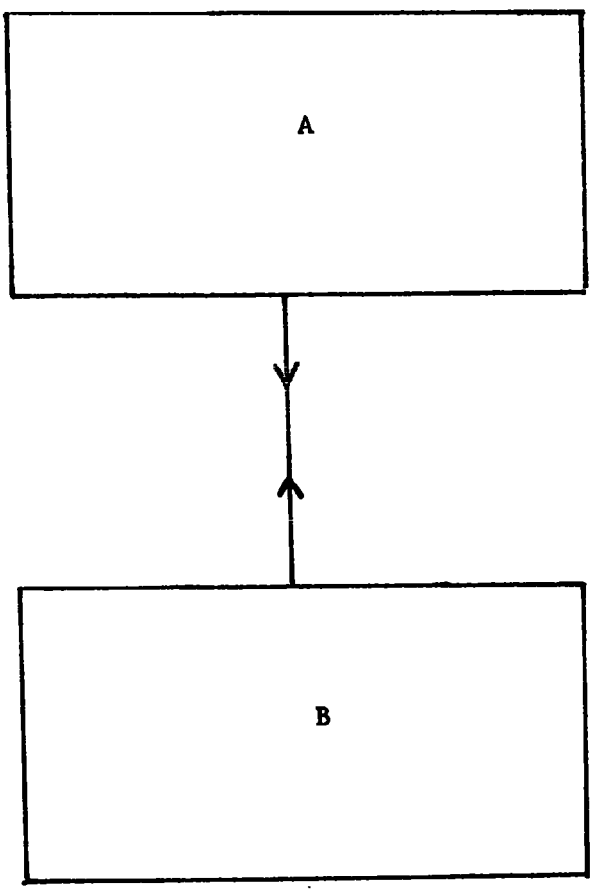


FIG. 2.2. ONE-TO-ONE RELATIONSHIP.

Simple - Complex Relationship.

Where one data element A relates to several of elements B, better known as a ONE-TO-MANY RELATIONSHIP e.g. a teacher relating to his group of pupils. The diagram (FIG. 2.3.) used in this instance.

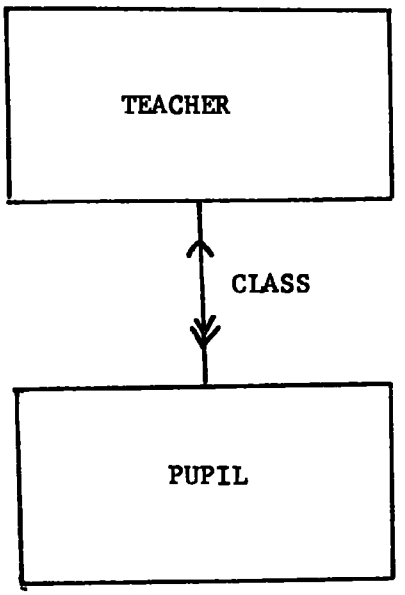


FIG. 2.3. ONE-TO-MANY RELATIONSHIP

Complex - Simple Relationship.

Better known as MANY-TO-ONE Fig. 2.4. describes this relationship. Once again using the teacher/pupil relationship, a one pupil class which has several tutors (a child of Victorian aristocracy?) is an example of a MANY-TO-ONE RELATIONSHIP between the same data elements as in Figure 2.3.

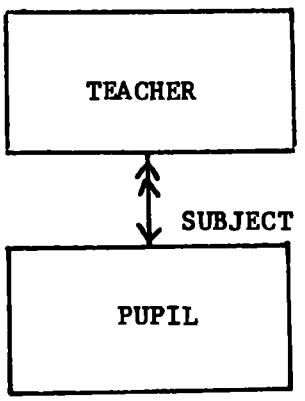


FIG. 2.4 MANY-TO-ONE RELATIONSHIP.

Complex - Complex Relationship.

It is better known as MANY-TO-MANY. The school timetable is an example. The teachers teach a subset of the school pupils in a class and each pupil has several teachers each one teaching a separate subject. This relationship can be seen as a combination of the 'class' and 'subject' relationships above.

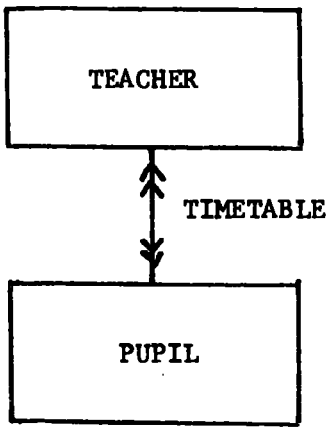


FIG. 2.5. MANY-TO-MANY RELATIONSHIP.

2.2.4. Entities and Attributes.

In section 2.2.2. the different data groupings were described. Such groupings are biased towards the way data is processed and stored but there is a need to describe the information structures of an organisation. The objects which describe information are called Entities and Attributes.

An ENTITY may be a tangible object, e.g. a part, place or employee, or a non-tangible object - a job title, customer account or abstract concept. Often we are concerned with a collection of similar entities such as employees, a group of entities is called an ENTITY SET. The entity is equivalent in data terms to the programmer's record.

An entity can have several ATTRIBUTES which describe the entity. Attributes are equivalent to the data fields in the programmer's record. One of the attributes identifies the entity.

The third concept of information is the ATTRIBUTE'S VALUE.

An example of the entity set is given in figure 2.6. Each row is an entity, each column is an attribute and one of the attributes identifies the entity (row). Each entity is a member of the entity set because each entity has the same attributes.

NAME OF ATTRIBUTE	EMPLOYEE NUMBER	NAME	SEX	DEPARTMENT	TITLE	SALARY
VALUE OF ATTRIBUTE	173654	WALTERS B.	1	165	PLUMBER	2500
	169548	JAMES K.	0	106	CLEANER	1800
	179548	WILSON R.	1	151	FITTER	2600
	184179	BELL T.	1	107	WELDER	2650
	ENTITY IDENTIFIER			SOME ATTRIBUTES ARE IDENTIFIERS OF ANOTHER ENTITY		

FIG. 2.6. THE ENTITY SET.

Using the three information concepts entity, attribute and value any element of information can be conveniently depicted as a point in a three dimensional space. The axes represent a set of entities, a set of attributes and a set of values. The intersection of an entity, an attribute and a value determines an item of information.

Many points in the information space are not relevant to an organisation, those that are relevant are provided by the concept of a data base.
A subset of points in the total information space is known as a DATA BASE.

2.2.5. Summary of the Nature of Data and Information.

There are three realms of data and information. There is data which is stored on computer files and accessed by computer program. Data records are a collection of data elements but all such elements are built upon a single data unit called the data item each item can be associated with a data item value.

Information is represented as a set of attributes with their attribute values describing the entities of the organisation. Information is provided by the data elements and by how these elements are related to one another, and further information is provided by the characteristics, properties and extent of these relationships.

The attributes represent the properties of the real world entities, the employees, parts and accounts etc. that are relevant to the particular organisation. In this way, the real world is described as a subset of the information space. The data and data relationships which represent this information are held in a data base.

2.3. DATA ORGANISATION.

The way data has been organised has changed as different storage media have become available, indeed the way data is viewed by the programmer has changed from the way data is viewed by the machine. The number of ways data can be accessed also greatly increases as more complex organisations are used.

2.3.1. The Card File.

The first storage media used was the card file.

Each card in the file is read into the machine and passed individually to the computer program with the data as viewed by the program, THE LOGICAL DATA STRUCTURE, identical to the data on the card, THE PHYSICAL DATA STRUCTURE.

2.3.2. Characteristics of Magnetic Tape.

The first aspect of storage medium is whether such a medium is off-line or on-line, An OFF-LINE storage device is one which requires human intervention before being accessible by the computer e.g. the loading of a magnetic tape. An ON-LINE device is one which requires no human intervention.

A magnetic tape is a serial device, meaning that all data must be read from the beginning of the file until the data required is found.

Magnetic tape supports only SERIAL FILE ORGANISATION and because access to data is by reading every record from the beginning of the file the ACCESS METHOD is SERIAL. The unit of processing is the record, however there is an advantage if the machine transfers more data than the program requires because if, for instance, the program requires a record of 125 words from the tape it would be more efficient for the machine to transfer 1 block of 500 words of data (i.e. 4 x 125 word records) than to do 4 transfers of 125 words, particularly since serial processing is implied by the type of access and organisation.

The programmer is not aware of the transfer of these 4 records and there is a need for data handling software to 'de-block' the records and pass each record to the program when required.

This is the first fairly trivial example of a difference between the programmer's view of the record as the unit of transfer and the machine's view of the block as the unit of transfer.

2.3.3. Characteristics of Direct Access Devices.

The direct access device provides much more flexibility in the methods of access by being able to go straight to the address of the data required. The device's read heads are positioned at the address in 2 combined movements:-

(1) The read heads move across the radius of the circular disc surface while,

(2) the revolving surface brings the particular data under the read heads.

Because of the nature of the device, several different file organisations (the way records within a file are placed) are permissable and several different access methods are possible (the way records in a particular file organisation are obtained for processing).

Of course serial file organisation and serial access are possible but normally a serial file would be stored on the much cheaper magnetic tape medium.

Some of the basic file organisations and access methods used on direct access devices during the 1960's and indeed up to the present time are as follows:-

Sequential File Organisation.

Magnetic tape files if updated are written to a different tape, but with direct access files it is possible to read records and update them writing the records back to the same file. The files can have their updated records written back with more fields included, increasing the record's size and whole new records can be inserted. To enable records to be inserted in the correct key sequence, should there be insufficient room, necessitates a pointer to be inserted in the correct physical location pointing to a location in an empty area of the file where the complete record can be stored, such an area is called an overflow area. The fact that records are obtainable in key sequence without being able to keep all records in physical sequence, designates the organisation as sequential.

The Sequential Access Method.

This gives the capability of obtaining the next logical record whether that record is physically the next record or whether the record is in overflow. Here once again, the difference between the logical view of records in sequence, and the physical view, which accomodates the limitations of the storage device are highlighted.

The Indexed Sequential Access Method.

Provides an access method which enables individual records to be accessed by following a table of pointers which point to every home (non-overflow) location on the file.

The Direct File Organisation.

This organisation places records at a certain location within the file by means of an algorithm which computes the result using some manipulation of the key. Overflow will be caused by the algorithm generating too many 'same-value' locations for different keys. Any overflow caused in this type of organisation is usually dealt with by placing such a record into the next available file location, or in more sophisticated systems, by placing overflow records in 'pools' of overflow areas at the end of the file or at convenient intervals within the file.

The Random Access Method.

Because the user does not know where a record will be placed within the file, serial or sequential access is not possible. The only possible method of access is random, that is, access is by simply applying the same algorithm that was used initially to put the record on the file.

Certainly in the 1960's the above access methods provided the spectrum of file access, from the area of batch processing with serial or sequential access to on-line single record transaction processing using random access. Usually the best solution is the compromise of indexed sequential access although there are occasions when the decision to use this access method is taken automatically when a better understanding of the problem would reveal a more appropriate file accessing solution.

2.3.4. Physical Organisation.

In the previous section file organisations and accessing were considered. Physical file organisation has many varied characteristics and it is worthwhile looking at several of these characteristics at this point.

Blocking.

As we have seen it is more efficient in storage space to block records together into a PHYSICAL RECORD i.e. a magnetic tape block. There is an added efficiency in processing should several of the logical records within the one physical record be required for processing in the serial processing situation. Clearly a decision must be made as to how many logical records should be packed into one physical record.

Variable Length Records.

In many cases logical records can be variable in length while the physical record length is a fixed size. Any spare area in the physical record is 'padding'.

Sequencing of Records.

In the previous section the concept of the programmer's logical record sequence being different to the physical sequence because of overflow was discussed.

Data Compaction.

Blocking has been described as a method of space saving, however there are many techniques available for compacting data, that is by storing data in a coded form rather than the normal storage form. The code could be found from a table or generated by an algorithm, it could be application dependent or application independent.

Minimisation of Redundancy.

DATA REDUNDANCY is the storing of the same data more than once. It is a characteristic of data base management system that they reduce redundancy, but such a facility is not a preserve of such systems, it can be used in any physical file organisation. Should a particular data field take one of a set of values, the actual value could be stored in a table and all fields taking that value could have a pointer to the correct entry in the table. This is useful should the value be regularly changed and in such a case only one data field will be updated.

Type of Processing.

As we have seen, physical organisation is dependent on the type of processing. The payroll type of application is best suited by a sequential type of processing consequently imposing a sequential file organisation approach, but other types of applications like stock control may have an on-line updating requirement best suited by a random accessing approach which may only be possible by using a random file organisation.

File/Activity Ratio.

A ratio between the number of records processed in a run and the number accessed in a run. If the percentage is high it may be advisable to use a serial organisation with magnetic tape as the storage medium, otherwise an indexed sequential organisation, or, if the ratio is very low, a random organisation. This characteristic is only worth studying in the batch processing application where this ratio could have a wide range, the transaction processing application will (or should) without question have a very low file/activity ratio per transaction.

Frequency of Reference.

Different data has different frequency of reference, 10% of the data is usually referenced 90% of the time while the other 90% of the data is accessed 10% of the time. The most frequently accessed data should be placed in the most advantageous position on the storage medium.

Response Time Requirements.

The placement and organisation of data should also be dependent on the response required.

Throughput.

Related to the previous 2 characteristics, the organisation of data should enhance the general throughput of the system.

Data Volatility.

A volatile file is one which has a high proportion of additions and deletions. A physical organisation should be chosen which minimises record addition problems. A static file using an indexed sequential organisation will eventually place additions in overflow areas making the accessing of an increasing number of records inefficient. The file must be reorganised to return it to its previous optimum state. The method of reorganisation should be as efficient as possible.

Clustered Insertions.

Related to the problem of date volatility is the need in certain circumstances to be able to insert maybe 100 new recordsin one location on the file. Not all file organisations would be able to permit this and still provide a reasonably efficient access to all records on the file.

Expandability.

If there will be more additions than deletions will the file organisation chosen be able to handle that situation?

2.3.5. Summary.

The different characteristics of the storage media used in the last twenty years have been examined, and in looking at the different media, the concepts of how a file is organised and how files can be differently organised and data differently accessed. Finally the different processing constraints and requirements which might influence the way data is organised have been discussed.

2.4. LOGICAL DATA STRUCTURES.

LOGICAL DATA STRUCTURES are the programmer's view of the data that he is processing, the COBOL record descriptions and data definitions which give the picture of the data that is to be processed.

In the late 1960's, early 1970's there were attempts at generalised packages based on certain types of logical data structures. The processing to handle such structures were built into the package and the user only had to provide the parts of the data structure and processing that were specific to the user's individual requirement. The ICL NIMMS production control package is such a suite of generalised programs which provides a batch update and reporting facility for the production control systems spectrum - stock control, requirements planning, shop floor scheduling etc. The suite is based on a hierarchical logical data structure, and generalised programs permit this structure to be updated and enquired upon. The user can easily describe the exact structure he requires, and the specific update and enquiry processing can be supplied by him as a subroutine to the generalised program. The accessing of data from the storage medium and the collecting of the data into the data structure the user requires is all done by the program.

In the early 1970's such a system was the first attempt at a data management system. It has 3 important advantages over the wholly produced user systems:-

(1) The way the data is stored is different to the way the user sees his data in the logical data structure. Consequently the way the data is stored can be best suited to the storage medium.

(2) The accessing of the data and the transforming of the data from the way it is stored to the way it is viewed is handled by the generalised program and need not be the concern of the user who only deals with his own logical data structure.

(3) Arising from (2), because most of the complexities such as data accessing are programmed within the product and written only once, the users development times should be much reduced and as a result the amount of maintenance should be similarly reduced.

This type of data management is best described diagramatically (Fig. 2.7.).

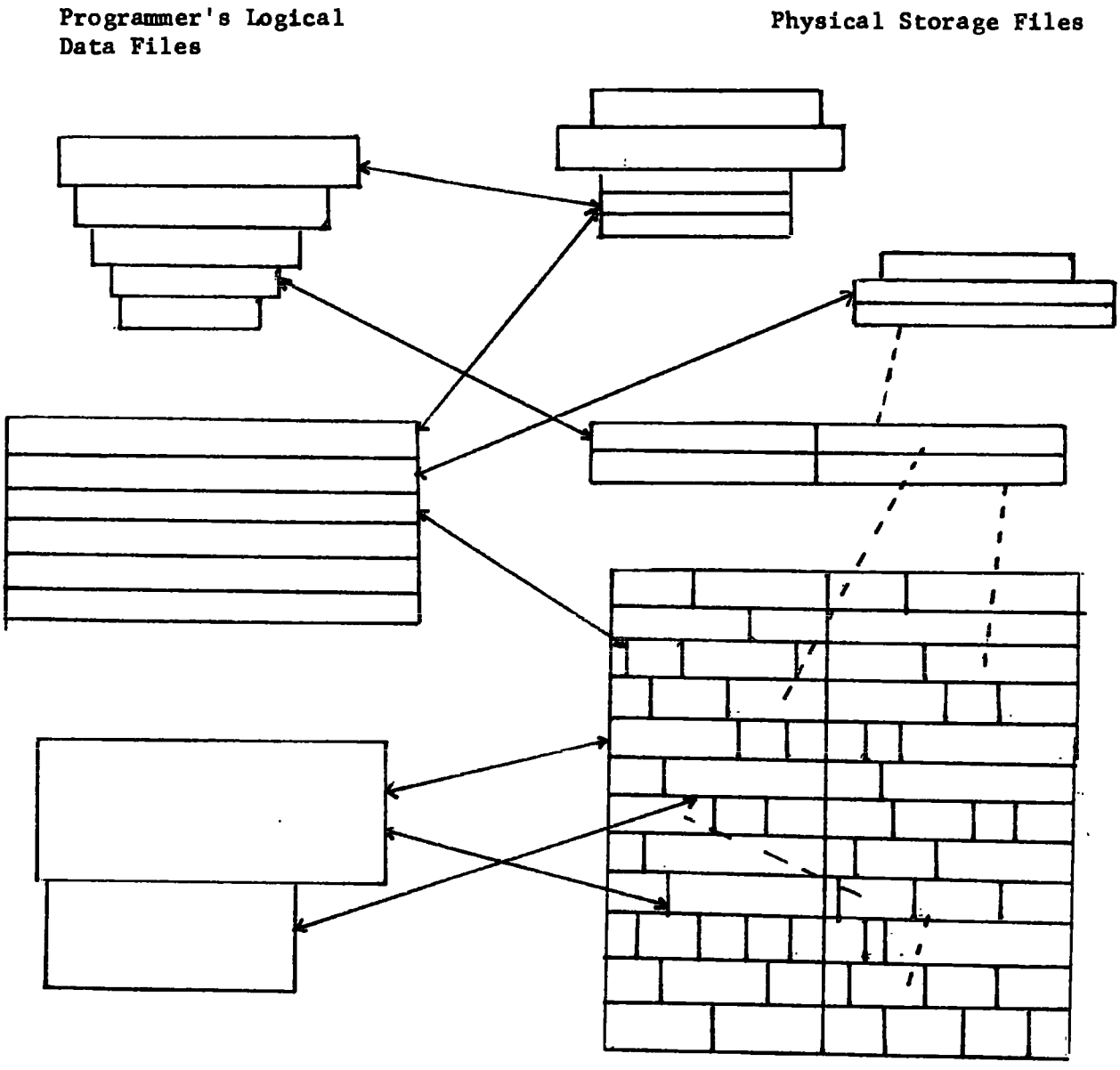


FIG. 2.7. EARLY DATA MANAGEMENT SYSTEMS.

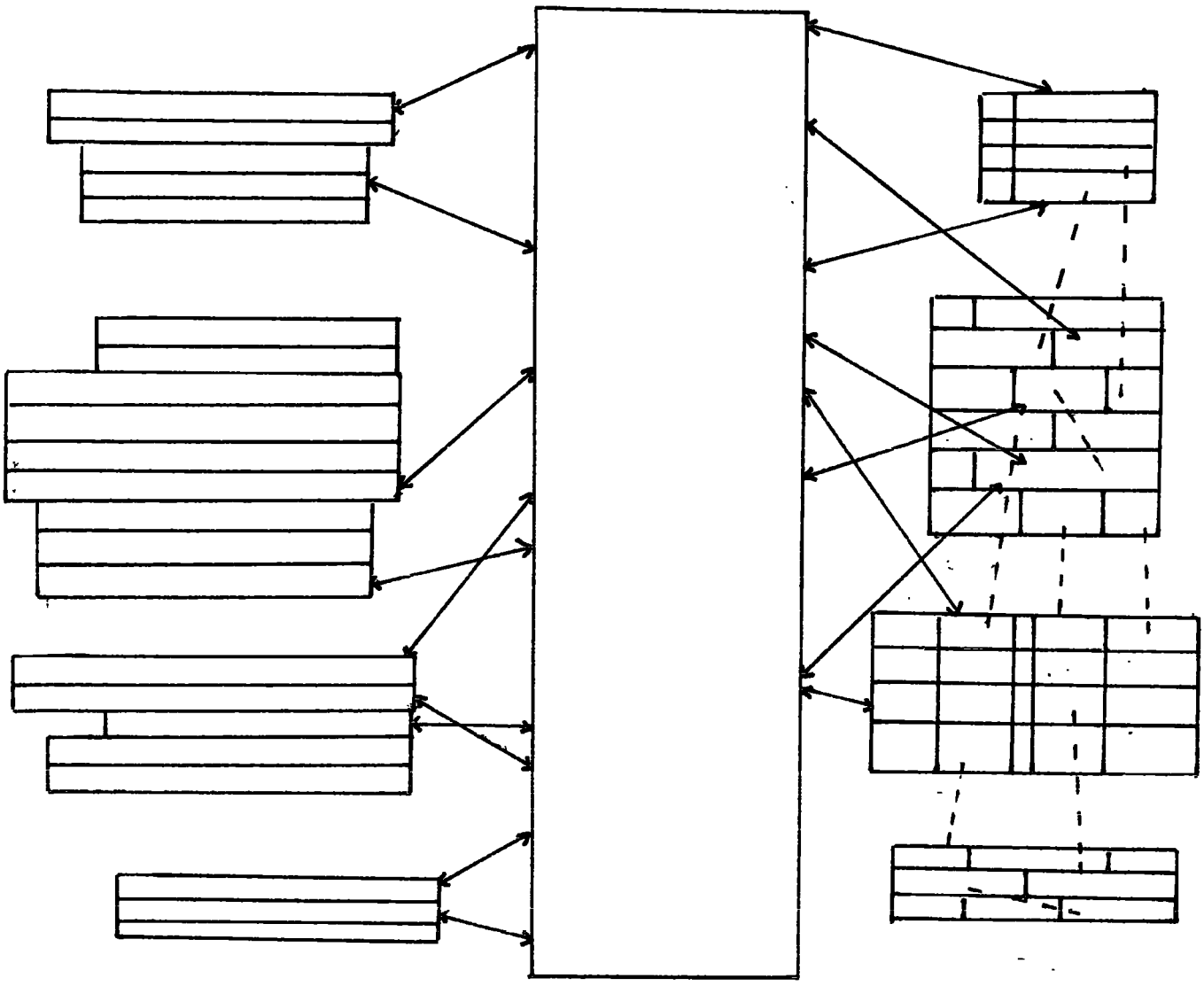


FIG. 2.8. TODAY'S REQUIREMENTS IN DATA BASE SYSTEMS.

The description of how the programmer's logical data structures relate to the machine's storage structures is called a MAP.

Many of what are termed the early data base management systems are based on this 2-structure approach. However this is a restricted approach

as it does not relate to what one might call the organisation's information model. Fig. 2.8. gives a picture of this 3-structure approach which is the requirement for today's data base management system, and is the approach on which this thesis is based.

2.4.1. Data Base Design.

The 3 parts of this design methodology are supported by 3 separate data descriptions:-

(1) The SUB-SCHEMA.

Or application programmer's logical file organisation, is the description of the needs of one or more application programs.

(2) The SCHEMA.

The global logical data description of the entire data base.

(3) The Physical Data Base Description.

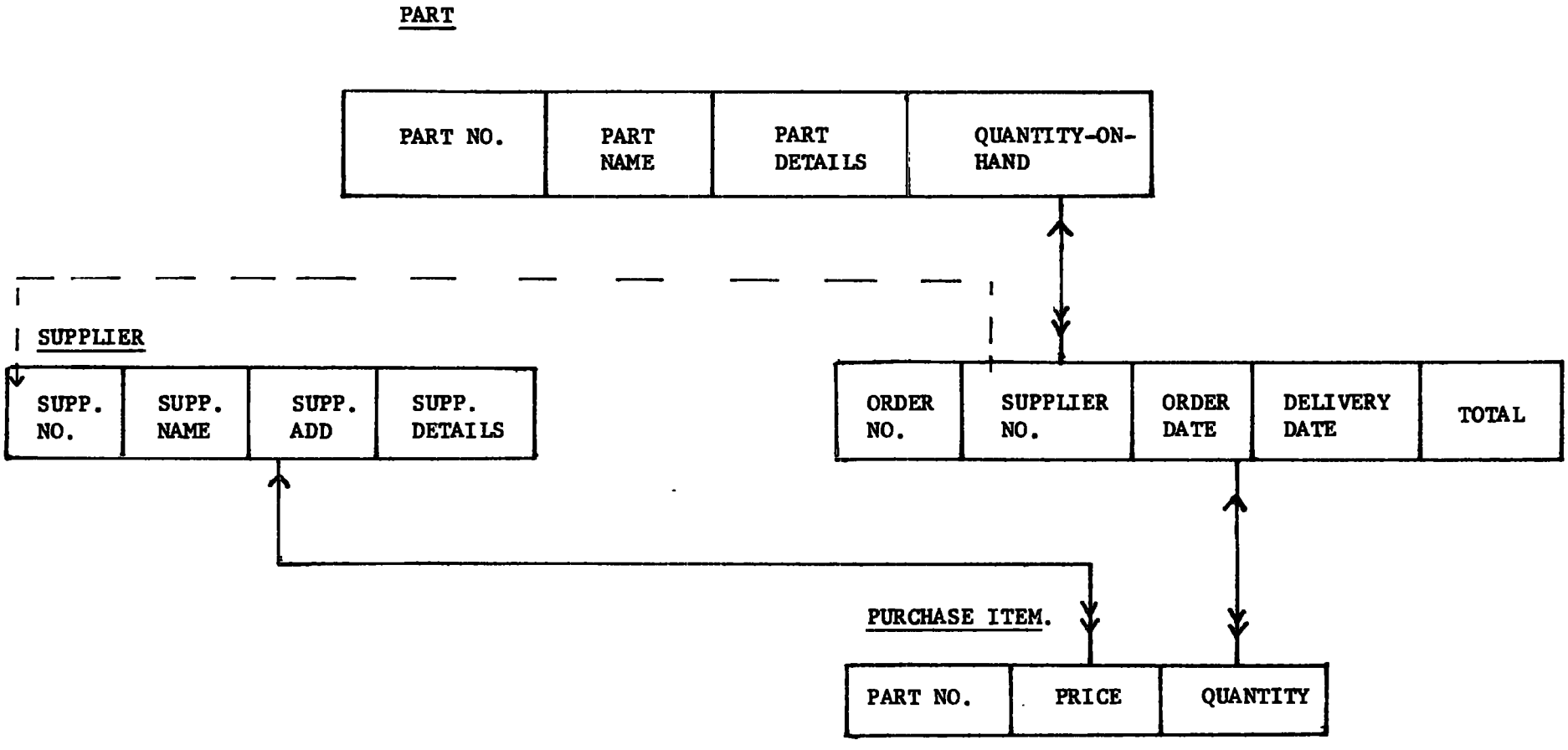
A description of the way the physical data in the data base is organised. It describes how data is located on the storage devices and the description can be altered to improve data base management performances. This particular description is dealt with in section 2.5.

The advantage of describing a data base in 3 different ways is the ease in which new applications can be added to use the data base, and in an evolving technological world, to easily introduce new data storage hardware.

2.4.2. The Schema.

The schema is the central description and is the most important aspect of the data base design. It directs how data is stored on the storage device and it reflects which data is required by all the D.P. applications using the data base.

It describes how the organisation's information requirement can be derived from the data base. The 'data groupings' which represent the real world entities are called 'record types' each having several data element types representing the attributes, one of which uniquely identifies



2.9. A SCHEMA

Describes the relationship between different record types containing different data item types.

the particular record and which is referred to as the primary key, equivalent to the entity identifier described in section 2.2.4.

When the data element types are paired with specific data element values the schema framework becomes an instance of the schema. When talking about records, it must be clear whether a specific record with values is intended as in the instance of a schema or the record type description which is included in the framework schema. It is important to be aware of the difference. The remainder of this section will talk about the schema framework.

The schema as the map of the organisation's real world data relationships, consists of record types linked by specific data relationships. The description of how records are related and the properties of these relationships are as described in section 2.2.3.

Such relationship's are described diagramatically by solid lines connecting the rectangular boxes which represent the record types. The boxes are sub-divided to show the data element types. There is a further relationship, this time between data elements within the records. A particular data element can occur in different record types and to indicate the fact that at the instance of a schema, the data elements have not a value stored in the record but a pointer to the record containing this data element's value, a dashed line is used to connect the two data elements with an arrow pointing to the data element which has the value. This dashed line is called a cross-reference line and it indicates linkages that exist in the physical data base to enable data to be found more quickly.

An example of a schema diagram is given in figure 2.9. The schema is made up of several different types of data structure.

Simple.

All data units are independent of each other and logically have equal significance.

Hierarchy.

Data units are dependent and can be arranged so as to have one unit at the level above in the hierarchy and/or one or more units at the next level down in the hierarchy.

Network.

The units of data are dependent but a data unit can have more than one other data unit directly relating to it from the level immediately above.

The implementation of the hierarchy and the network are called the TREE and PLEX structures.

2.4.3. The Tree.

A tree structure is a hierarchy of elements, called NODES and the uppermost level in the hierarchy has only one node, called the ROOT NODE. With the exception of the root node, every node has ONE AND ONLY ONE node related to it at a higher level. Each node can have more than one node related to it at the lower level. The nodes at the base of the tree have no nodes at the lower level and are called LEAF nodes.

Trees can be used in any of the three data descriptions mentioned in section 2.4.1. and the nodes can represent anything from data items to record types.

The tree structure is a recursive structure and Knuth uses this fact in his definition of a tree as "a finite set T of one or more nodes such that:-

(1) There is one specially designated node called the root of the tree.

(2) The remaining nodes are partitioned into $m \geq o$ disjoint (i.e. not connected) sets $T_1 \ldots T_m$ and each of these sets is in turn a tree. The trees $T_1 \ldots T_m$ are called the subtrees of the root ".

The Binary Tree.

The binary tree is a special kind of tree structure permitting up to only 2 branches per node. A typical example of a logical data organisation as a binary tree is a dog's pedigree.

The data relationships described in section 2.2.3. show how the data elements of any such structure relate together. If the next highest node is regarded as the 'parent' node and the next lowest level is regarded as the 'child' node, the mapping from child to parent is a simple mapping

and the inverse mapping from parent to child is complex (one-to-many). Occasionally there is a one-to-one relationship which usually means that the tree structure relates to records of the same record type that are stored separately.

Hetrogeneous and Homogeneous Structures.

A heterogeneous tree structure is one which relates different record types together. Most data base systems are designed to handle the heterogeneous trees of fixed depth, whereas the homogeneous tree structure defines different data relationships of possibly variable depth with regard to data of the same record type. For instance the way different parts relate in a bill-of-materials data structure is a homogeneous structure because all data is of the same record type 'part'.

2.4.4. The Plex Structure.

A PLEX (or NETWORK) is a structure which has a data element which can have more than one data element relating to it at the next higher level. Any data element can be linked to any other data element in any way in such a structure.

Simple Plex Structures.

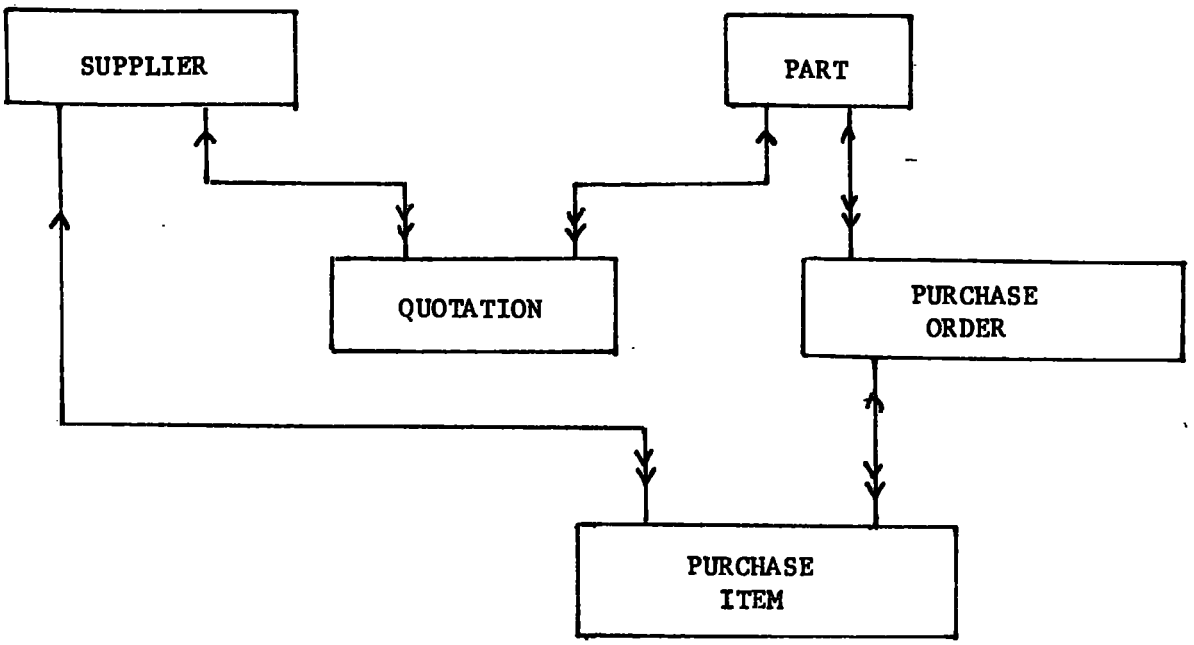
Figure 2.10. shows a heterogeneous plex structure consisting of five record types. It is a simple plex structure because the higher-level to lower-level relationship (parent-to-child) is complex while the child-to-parent relationship is simple. The same pointer conventions are used as described in section 2.2.3.

The definition of a simple plex structure is one in which no line has double arrows in both directions.

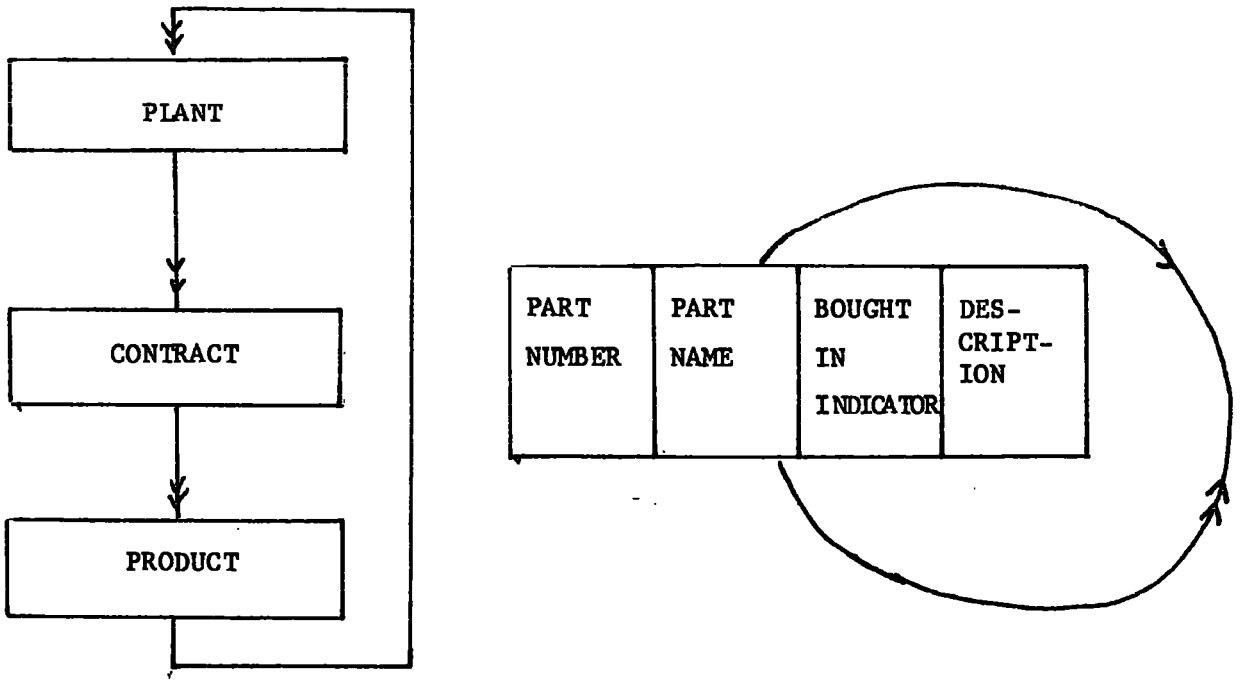
Complex Plex Structures.

The data structure which has a relationship between two data types that is many-to-many is a complex plex structure.

There is a significant difference between the simple and complex plex structures because the latter structure requires more elaborate methods of physical representation and quite a number of data base management systems cannot handle such structures.



2.10. A SIMPLE PLEX STRUCTURE.



THE CYCLE

THE LOOP

2.11. THE CYCLE AND LOOP.

2.4.5. Cycles and Loops.

A CYCLE is a structure where a data type has as its descendant an ancestor of this data type. The relationship between the 'parent' data type and 'child' data type is a cycle structure.

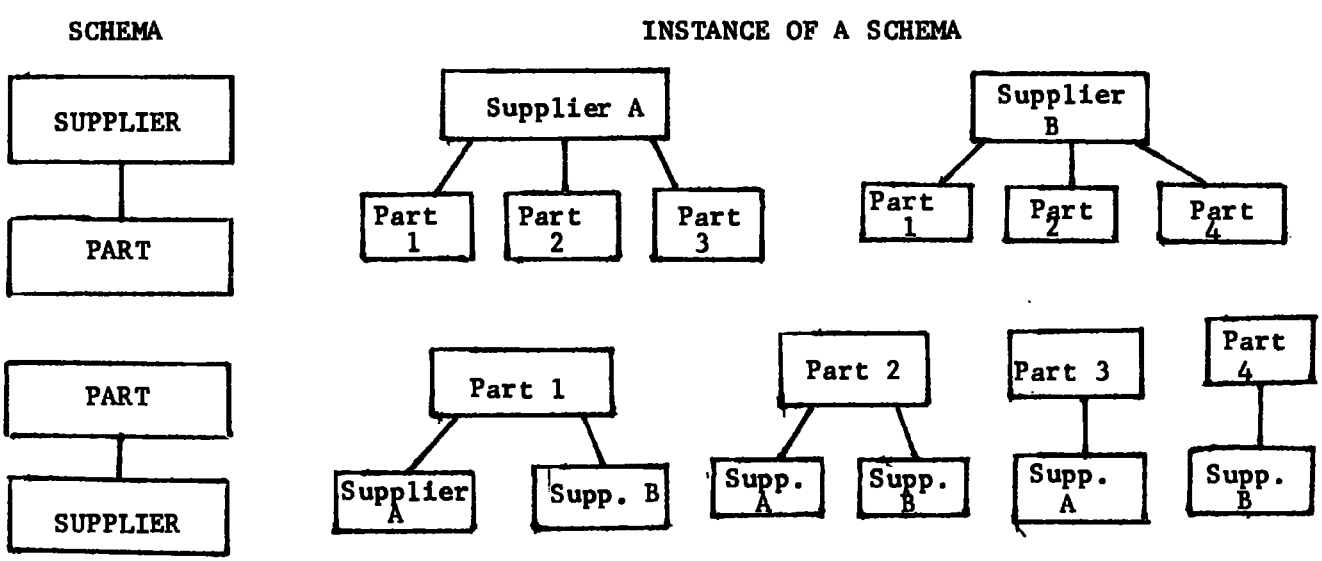
The homogeneous tree structure which defines how data elements of the same data type are related, can be represented as a LOOP. The structure defining how 'parts' in a product relate to one another would be represented by a loop structure in the schema.

Figure 2.11. gives examples of the cycle and loop structures.

2.4.6. Plex Representation.

A plex structure can be represented by reducing it to a simpler form by introducing REDUNDANCY, that is duplicating certain of the logical data. The redundancy may be small and tolerable or excessive and intolerable. Such redundancy does not imply physical record storage redundancy but rather a duplication of the logical view in the schema.

Complex plex structures are usually difficult to break down into a simpler form. In general each complex relationship needs to be replaced with two tree-structured links as in figure 2.12.



2.12. A PLEX STRUCTURE REPRESENTED IN A SIMPLER FORM.

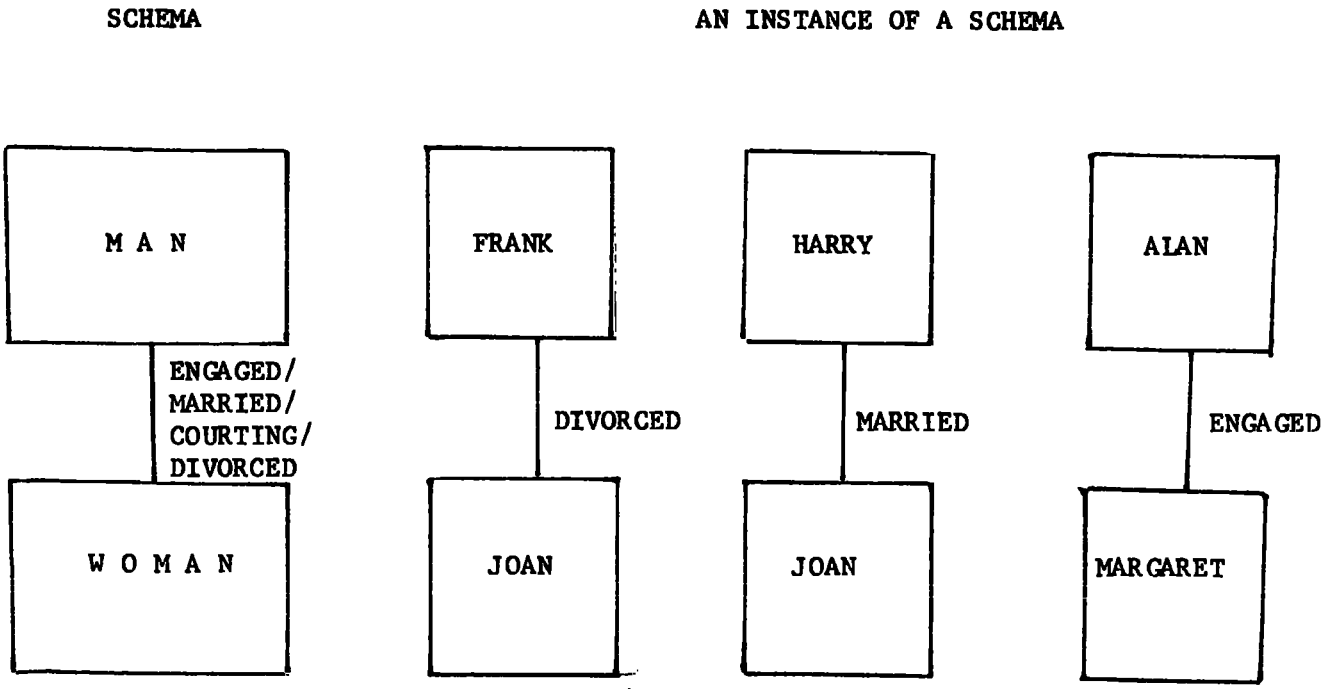


FIG. 2.13. MULTIPLE RELATIONSHIPS.

2.4.7. Schema Multiple Relationships.

There can be more than one relationship between two data types. Figure 2.13. shows that there are several relationships possible between a man and a woman. At the instance of a schema, however, there will be two data elements related by one relationship.

2.4.8. The Sub-Schema.

The schema is an overall chart of how data element types relate to one another. A sub-schema represents the application programmer's view of the data element types he uses.

The way the sub-schema and schema relate is analogous to the idea of a road map. The schema is equivalent to a full road map, it maps the data base, while the sub-schema gives the route on the map along which we wish to travel, in other words it is the route through the data base that obtains the data and presents it in the form required.

The sub-schema is derived from, is a subset of, the schema. The schema completely describes the data base, the sub-schema describes the data requested.

There is a need to understand and define the user requirements for the sub-schema. R. Durchholz and G. Richter described the concept of 'Constructs' (ref. 1).

2.4.9. Constructs.

The way the user wishes his data to be organised is not just dependent on the 'real world' structure but also on the way the user wishes to process it. A clear picture can be achieved if the data organisation and data handling are defined.

The data organisation of the sub-schema is a selection process, a reduction of information and the imposing of a certain structure on that information. This model information described by the sub-schema is termed a CONSTRUCT. By looking at the different CONSTRUCTions it becomes clear what structuring facilities can be provided for the application programmer.

All constructs are built from the elementary construct, the value. The picture can best be drawn by using boxes (Fig. 2.14).

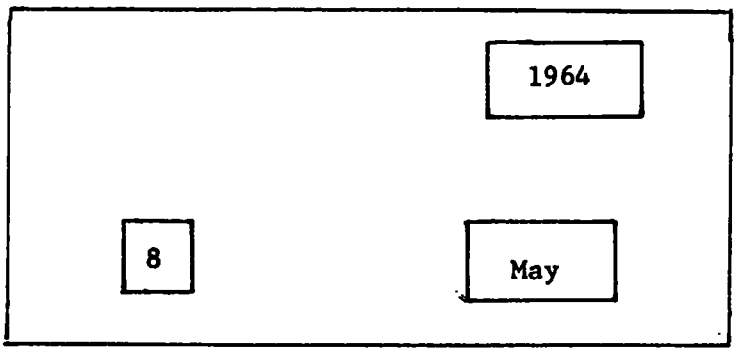


FIG. 2.14.

To aid the selection process, labels or names are added to the values. They are relevant only to the sub-schema and are the means of defining structure (Fig. 2.15).

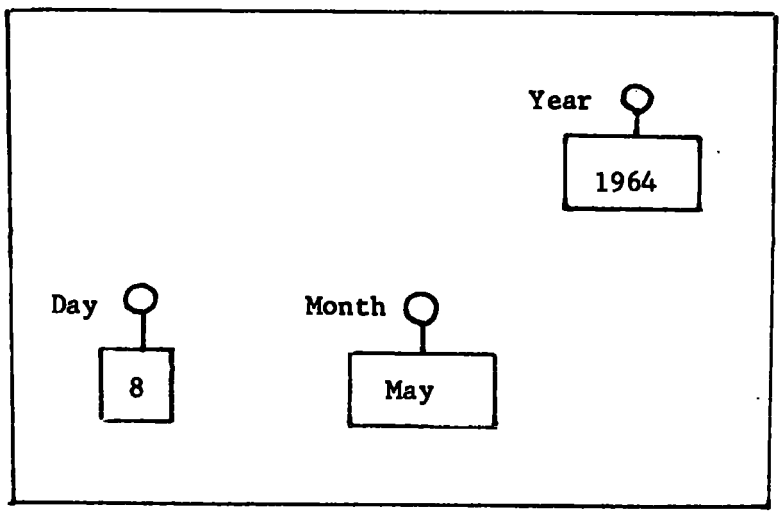


FIG. 2.15.

There are 2 types of aggregates:-

Nominations.

Nominations have named constructs as constituents.

Collections.

Those aggregates having unnamed constructs as constituents.

Nominations can be described as the mapping of names on to constructs. Collections can be described as an unordered finite set.

If there are several levels of aggregates the name which is used as the identifier must be qualified by using the names of the higher level aggregates. A name is only a simple identifier.

Figure 2.16. provides the diagramatic representation of what has been described.

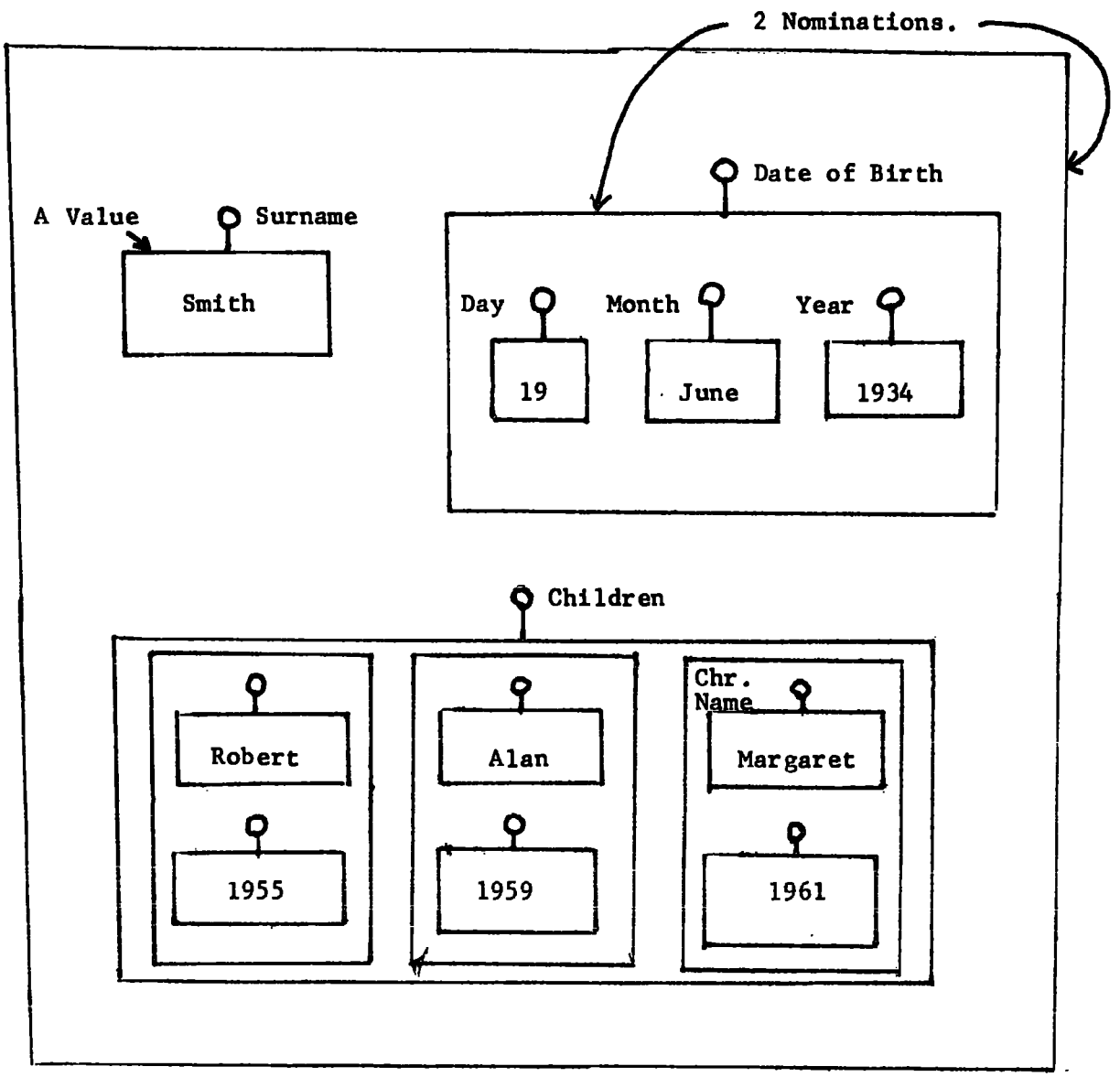


FIG. 2.16.

A construct is only a model of the set of information and as such can be a constituent of another model. To this end it is worth remembering the following points:-

- one construct can be a component of many constructs;
- one construct can be a component of nominations and collections at the same time (i.e. in different contexts a name can be attached or not);
- a construct can have different names (in the same or different context).

The user may have all these features in his mind when forming constructs from his knowledge of the 'real world' (the schema).

A type is described as an equivalent to record type. The name of the type is put in a small box in the right hand corner of the construct symbol.

This 'type' concept enables the idea of a repeating construct. Figure 2.17. shows this idea with the 'project' and 'department' repeating constructs within the 'organisation' type.

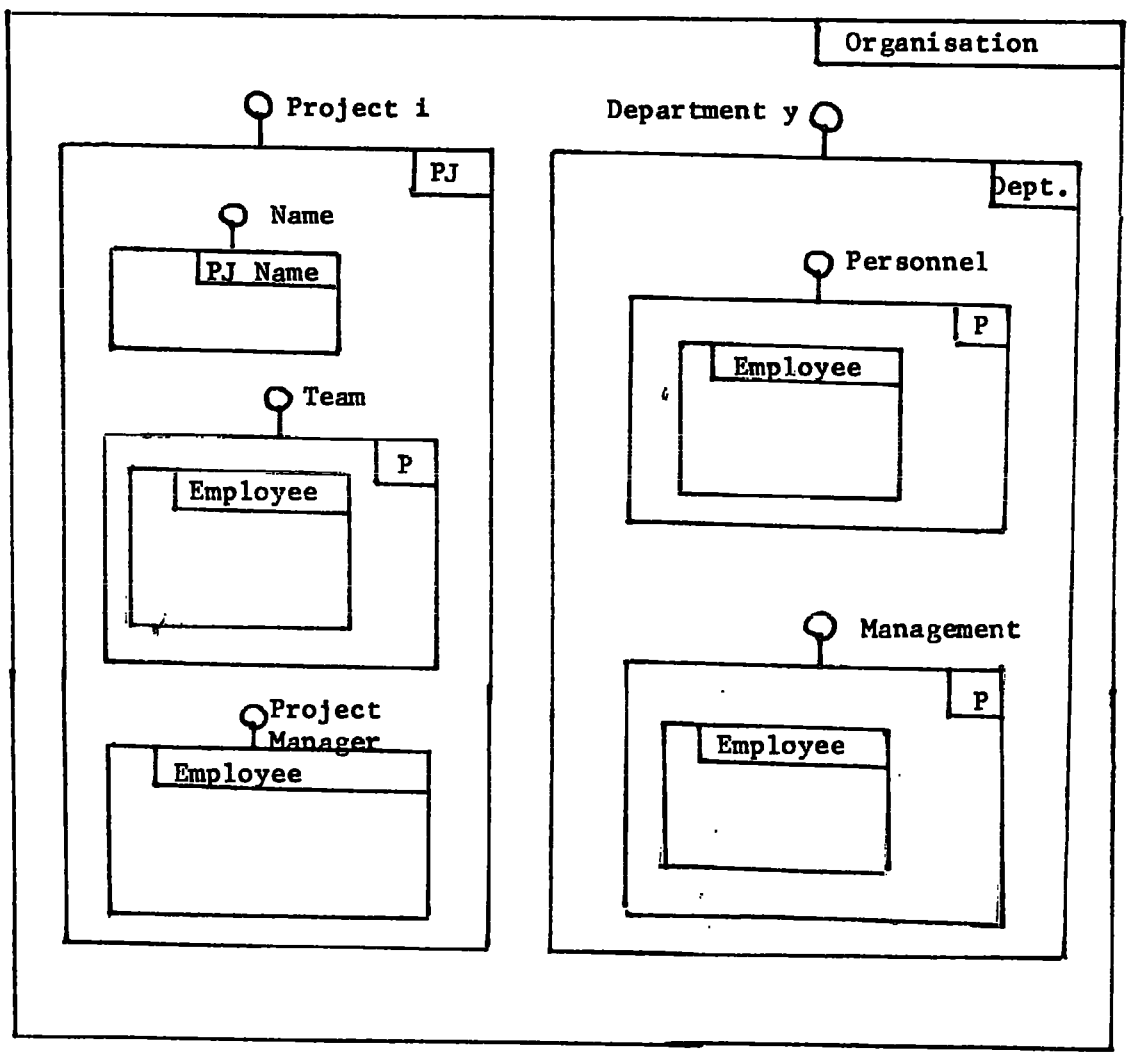


FIG. 2.17.

The 'project' and 'department' constructs are 'types' in their own right.

The difference between construct and type is that the type expresses a restriction of the universe, that is, an agreement with respect to the area of communication between user and data base, whereas the construct is a model of the information.

2.4.10. Information Handling with Constructs.

At this point the authors introduce the idea of the 'framework construct'. This particular construct is useful as it is regarded as the all-embracing aggregate containing all the constructs the user requires as components. Clearly, the important point concerning this user representation of data is in the interface between user and data base and how data is transferred.

It is possible to reach any point in the framework construct by using a sequence of names and properties, this point the authors describe as a spot. The user in his information handling deals entirely with spots. He is able to remember and refer back to them, and he is able to label them.

The manipulations for spots, suggested by the authors, are as follows:-

- on constructs

(1) input construct (to DBMS);
(2) output construct (from DBMS);

- on constructs at spots

(3) insert component (generate a spot);
(4) delete component (discard a spot);
(5) replace component (at a given spot);

- on labels

(6) attach label (to a spot);
(7) remove label (from a spot);
(8) shift label (from one spot to another);

- on spots

(9) lock spots (against access);

(10) unlock spots;

- on DBMS processes

(11) cancel (current execution);

(12) stop (current execution);

(13) continue (current execution).

2.4.11. Comments on the Construct Concept.

Possibly the most important aspect of this concept is that it highlights some of the important aspects of data structuring in the sub-schema. There are two aspects of the sub-schema described:-

(1) The idea of the need to model the information at the user interface; to build complex structures from the 'real world' model in the schema by using a basic 'building block' the 'construct'.

(2) Only a finite set of data is required, the idea that only a small subset of the data base is to be used and the need to define that range in the concept of the 'type'.

The ideas behind the concepts will be further discussed in part 2.

2.4.12. Summary of Logical Data Structures.

In the early 1970's data base structuring at the storage level and at the programmer's level had evolved into a 3-level system with the addition between the programming and storage levels of a central 'real world' information description which describes how an organisation's complete data resource is related, irrespective of the way data is accessed or stored. This schema, as the 'real world' description is known, has 2 aspects. The schema can either be referred to as a general description, or as the specific data at the instance of the schema i.e. when data resolution takes place.

Many different types of data structure can be used in the schema description. The characteristics of each of the data structures, trees, plexes, cycles and loops have been looked at briefly with special emphasis on the difference between homogeneous and heterogeneous structures.

The user will want fairly varied and flexible structures by which to process his data. The facilities provided by the construct concept are an attempt at describing the requirements of the data interface between the user processing and the data base management system, commonly called the sub-schema.

2.5. PHYSICAL DATA STRUCTURES AND STORAGE.

The way data is organised on storage devices is the largest single factor to influence the efficiency of access to the data base. The physical organisation must attempt to provide the best possible solution to the many different ways in which the data will be accessed. Of course the schema's global logical view, which sits between the physical data structure and the many conflicting sub-schema views, should have a narrowing effect on the choice of options for the physical data structure, but undoubtedly there still will remain difficulties in trying to provide both flexibility and efficiency in the accessing of the data.

The following sections describe some of structural and organisational choices open to the data base implementor.

2.5.1. The Basic Access Techniques.

The disc file organisations of the 1960's provide all the basic access techniques on which today's data base physical data structures are based.

They are as follows:-

(1) Accessing directly using some algorithmic function performed on the value of some data element.

(2) By indexing the value of certain data field(s) with the data elements address.

(3) By pointer, the address of one data record is held within another record.

Regardless of accessing technique, the address used to identify the storage location of data can take one of several forms.

2.5.2. Addressing Techniques.

There are three types of addressing:-

(1) The Address is the Device's Hardware Address.

This is the most efficient method, but has the great disadvantage that the address must be changed should the addressed data be moved.

(2) A Relative Address.

Each record is given a sequence number which gives the record's relative address within the file. This type of addressing is only slightly less efficient than the machine address, but it has the advantage of being shorter than the machine address, although once again the data record cannot be moved without changing the address.

(3) A Record Identifier.

The record identifier is usually converted to a relative address by either calculation or by an index as previously described. This method has great flexibility because it allows records to be placed anywhere, but without care this method can be very inefficient if the next record required is at the opposite end of the file.

This record identifier is a common term used in all data descriptions for both logical and physical records. It is a unique number or character string usually called a KEY. Sometimes more than one field makes up the key, in such a case it is called a concatenated key.

A particular record can have several fields as separate key fields. Each field can be used for a separate access path to the data. This type of key field is called a SECONDARY KEY.

One key must however identify the record, it is called the PRIMARY KEY.

The problem is how are records located with this key?

We have seen that scanning through the file serially may be too slow, but there are several different ways of searching for a record.

Block Search.

A block search looks at every 'nth' record and when the key is found that is larger than the one required, the search looks within the range of the previous 'n' records.

Binary Search.

A binary search looks at the mid-point of a search area and halves the search area, which half is dependent on whether the mid-point value is less or greater than the value required.

Indexed Sequential Search.

The records in the file on which this search is performed must be in key sequence order. A key is input to a table or index and the result of this look-up is a relative address or the actual address of the record sought.

Indexed Search.

An indexed search is a more generalised approach than the previous method, the table operates with a procedure that uses information about certain data field values and outputs information that will permit the record(s) with those data field values to be quickly found.

There are two types of index:-
primary index - is an index using primary key input;
secondary index - is an index using secondary key input.

If records are in key sequence there is no need to keep a table entry for each record but only for blocks of records.

Indexed Nonsequential File Search.

Such a search has to have an indexed entry for each record, consequently the search time is longer because the index is longer. This type of search must be available so that access can be made through the secondary keys which aren't in sequence.

Direct Addressing.

The record is located by transforming the key into the record's relative address. Key conversion algorithms are used which calculate the address of a record using the key value.

HASHING is the name given to the process of converting a key to an address, some of the conversion techniques are described below. There is always the problem of SYNONYMS when inserting records by this hashing technique. Synonyms refer to the keys that give the same address as keys with other values. The unit of addressing on disc files is usually called a PAGE. If there are too many synonyms, that is, too many records addressed to the same page, the home page, then there must be an overflow capability by either placing the record in the next physical page or in a page in a special overflow area.

2.5.3. Hashing Techniques.

There are several factors that can be changed in using a hashing technique:-

(1) the size of the page;
(2) the packing density;
(3) the key-to-address algorithm;
(4) the method of handling overflows.

If the page size is too small there will be more overflow, and if the page size is large, more time will be needed to transfer the data, because the page is the normal unit of transfer and it will take longer to search the data.

Some of the hashing techniques are as follows:-

The Mid Square Method.

The key value is squared and the central digits are adjusted to fit the range of addresses. The key may require to be converted into a digital form.

Division.

The key (after being digitised) is divided by a number close to the number of available addresses in the file. The remainder is taken to be the address value.

Shifting.

The outer digits are shifted in to overlap by an amount equal to the address and the result is in the address range e.g:-

Key - 16904778

1690
4778

6468

Folding

A technique similar to the folding of paper:-

19640782

287
691

978

Radix Conversion

Taking the key 172148 gives:-

$1 \times 11^5 + 7 \times 11^4 + 2 \times 11^3 + 1 \times 11^2 + 4 \times 11^1 + 8 \times 11^0 = 266373$

and truncate product finally multiplying by a constant to provide a value in the correct address range:-

$6373 \times 0.7 = 4461$

2.5.4. Chains.

Chaining is a technique in which records are linked together by pointers to permit a logical organisation to be derived from a physical organisation. This can lead to fairly flexible linkage between records within the same file and between records in different files. However the chains can become very long and very inefficient to follow. There is a need to know where a chain starts and ends.

Some data base packages have been designed on the basis of chaining the detail records to a master header record to enable logical structuring to take place. Certain optimisations can be made by putting the most frequently accessed records near the head of the chain, however this optimisation cannot be made if the chain expresses the sequencing of records.

It is possible to link the end record to the first record, this gives a RING structure.

The main problem in the use of chains is the search time to find the required record. The following variations on the chaining technique are attempts at reducing this search time.

Multi-list Chains.

The chains are 'cut' into segments and the head key of each segment is an index table entry. A record is quickly located by scanning the list of chain header keys and following the correct shortened chain.

Cellular Chains.

These chains are a special type of multi-list chain, the extent of the chain is within one cell. A CELL is usually defined as the largest area on an electro-mechanical storage device covered, which does not require mechanical movement of the read/write heads (the 'seek' area), on a magnetic disc this is usually a cylinder. The accessing of such a chain is an order of magnitude faster if the chain is held within such a cell than if the chain straddles a cell boundary.

The Parallel Cellular Chain.

The chain is a particular type of cellular chain. Each chain segment is held not only within a cell but also on different storage divices to the other segment chains which make up the whole multi-list chain, enabling the different segments to be accessed in parallel.

Extra pointers can be included in the chain to aid efficiency and recovery should failure occur during inserting records in the chain. The different chain pointers are as follows:-

Pointer from header record directly to the end record, this aids insertion at the tail of the chain.

A one way ring is a chain with a pointer from the end record to the header record.

A 2-way ring with both forward and backward pointers provides security should failure occur during the insertion of records.

Finally, a ring with pointers from each record back to the header record.

2.5.5. Physical Representation of Trees.

The relationships that are used to describe trees can use physical positioning, chain, index tables or bit maps showing the connections.

Trees can be of many different forms:-

A file that is not perfectly flat i.e. records consist of repeating groups within the records.

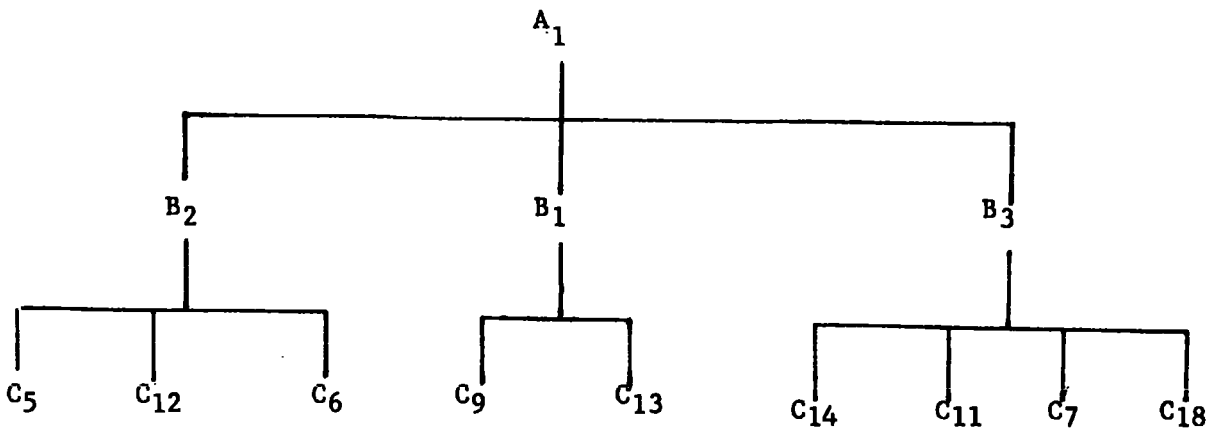
A common 2-level tree file with the transaction records at the lower level associated with one master record at the higher level.

There can be multi-level structures with different record types at different levels.

There are several different ways of representing trees each method having differing degrees of suitability depending on how the physical tree organisation is to be used.

Physical Contiguous List.

This was the first implementation of tree structures and it has the advantage of being able to be stored on a serial device. Figure 2.18 describes such a tree and how it would be stored. The ICL DMS 2 data management system uses this physical file organisation. The order is defined by navigating through the structure from top to bottom and left to right.



A_1 (B_2 (C_5 C_{12} C_6) B_1 (C_9 C_{13}) B_3 (C_{14} C_{11} C_7 C_{18}))

FIG. 2.18. A TREE STRUCTURE STORED SERIALLY.

If a direct access device is used for this file organisation on-line insertion of records into the tree structure becomes possible. This implies that the storage requirement for such a file may grow. This can be handled in two ways:-

- By using an overflow area in a similar fashion to its use in indexed sequential files.

- By the use of distributed free space. Areas of free space are allocated at points within the file. The most suitable place for the clusters of free space is at the end of each cell (seek area).

Pointers.

Many different pointers can be used to relate to the different records in a tree organisation.

Multiple Child Pointers.

Each record contains a pointer to all its children. This can mean a large number of pointers in the parent record should the parent have a lot of children.

Parent Pointers.

This is the inverse approach to the multiple child pointers, each child having a pointer to its parent record. In a tree structure with a child having only one parent, each record needs only one pointer field.

Child-and-Twin Pointers.

In this arrangement of pointers the parent record points to the first child record which in turn points to the next child record. This type of pointer mechanism can be used to construct chain and ring structures. Figure 2.19. gives an example of these 2 structures.

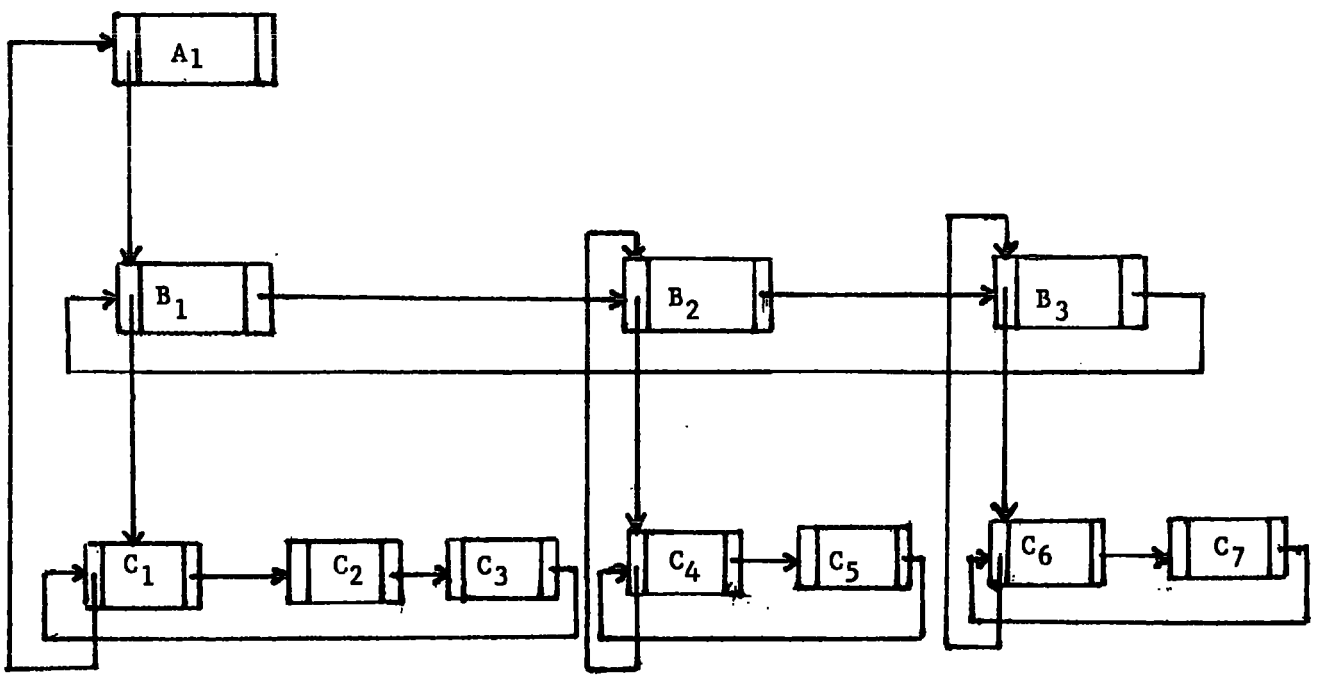


FIG. 2.19. CHAIN AND RING STRUCTURE.

2.5.6. Tree Directories.

There is no need to store the pointers which describe the tree structure in the data records themselves. Records containing the relationships between data records can be stored in a separate file called a DIRECTORY. The pointers that relate the data records into a tree structure are stored in a file called a tree directory.

2.5.7. Physical Representation of Plex Structures.

There are two types of plex structures (section 2.4.4.):-

(1) simple plex structures;
(2) complex plex structures.

The two different types are suited to different physical representations.

Physical Contiguity.

It is only possible to represent a plex structure by physical contiguity by storing data more than once. In figure 2.20. the 'C' records must be repeated.

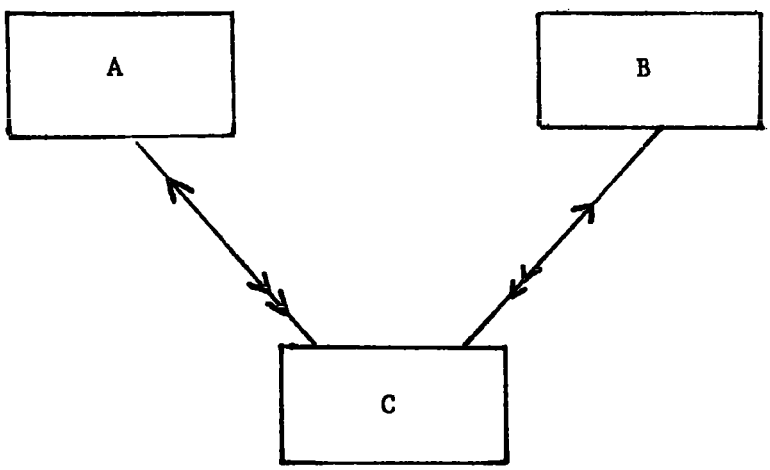


FIG. 2.20.

A mix of representations is possible if record 'C' is nearly always required after reading record 'A' but not always after record 'B'. It would be sensible in such circumstances to store 'C' records next to 'A' records and to refer to 'C' records from 'B' records by any of parent, multiple-child or child-twin pointers.

Insertions would of course force the use of overflow or distributed free-space mechanisms.

Complex plex structures can be represented but may require a large number of pointers to be stored and the capability for the pointer list to vary in length.

Multi-level plex structures have multi-level relationships. In structures there can not be assumptions concerning the level being pointed to. Indeed there is no obvious hierarchical structuring.

Directories.

Equivalent to tree directories, the pointers can be held separately. The directory records can be searched more quickly and they can be organised optimally and securely.

Bit Maps.

This is a mechanism to indicate which records are related. A '1' indicates the presence of a relationship between the record identifiers which are the co-ordinates to the map. It is a form of directory but they are usually only more spatial efficient than normal directories if the number of possible identifier values are within a limited range.

2.5.8. Multiple Key Retrieval.

The identifier of a record is normally referred to as the PRIMARY KEY This key usually identifies the location of the record on the file, but there is a tendency to place records for optimal access rather than maintaining record sequence.

Other data fields within a record can be used to identify a record for selection. This data field is called a SECONDARY KEY. Unlike the primary key, its value only indicates whether the record in which it is contained is to be accessed, it does not identify the record or (almost invariably) never indicates the records physical position. A record can have as many secondary keys as required, it can be made up of individual fields or concatenated fields, indeed any combination of data values.

Access through these secondary keys can be supported by any of the accessing methods that have been described. However, it is likely that within the data base files the same data can be obtained by several different access paths. To enable the access to be most efficient requires the data management system to know the length of chains, to ensure the shortest chain is followed.

2.5.9. The Inverted List.

This list organisation is the most common method of secondary index access. A list is kept of all key addresses that take a particular value, or range of values.

As an example, if an enquiry is to be made on the basis of the colour of a car. The data field 'colour' describes the record 'car'.
In implementation, an enquiry on the colour of cars means that the 'colour' field must be regarded as a secondary index. Should the question be asked 'which cars are blue?' A list would be kept for each possible colour and the 'blue' list would give the record addresses of all 'blue car'.
The primary key for each record would of course uniquely identify each car, probably the vehicle registration could be used.

Each entry in such an inverted list could be held as an INDIRECT INDEX. This means that the pointer to a particular record is indirect by storing the primary key value rather than the record's hardware address. This technique enables the address to be resolved later by a hashing technique or by a primary key/address conversion table. This gives more flexibility and reduces the data re-organisation problem.

If every field is a secondary index this can cause an expensive overhead in terms of list storage space. A method of reducing this overhead is to combine chaining with the list structure as described for primary index lists in section 2.5.4. A cellular multilist file structure can be used with the list entry pointing to a cell, with the records having the same secondary index value chained together within that cell. There would be no appreciable fall in performance if the cell size used is a convenient unit of transfer.

2.5.10. Index Searching.

It is worth taking a look at some of the list searching techniques.

Serial Scan.

Each entry is looked at.

Block Search.

The entries are divided into sections and the highest entry in each section is looked at from the lowest value section upwards. When an entry in the section has a value greater than that of the key required, the correct section has been found and this can be scanned.

Binary Search.

A common technique where the value of the middle entry in the index set is compared against the key value. If the key has a value greater than the entry the upper half of the initial set becomes the new set, otherwise the lower half becomes the new set, and by using this technique recursively the correct entry is quickly found. This technique is very fast if the index is in main memory but very poor if the index is spread over several direct access blocks or worse still over several seek areas.

Binary Tree Index Search.

A recursive technique using 3 steps to 'walk down' or PREORDER TRAVERSE the tree.

(1) Store the root of the tree.
(2) Traverse the root's left subtree in preorder traversal sequence.
(3) Traverse the root's right subtree in preorder traversal sequence.

This technique is very good for insertions or deletions but space is taken up to include the pointers.

Balanced Tree Index Search.

The search commences at the highest level node and works down through the levels to the data. Each of the nodes can be scanned by a serial, block or binary search.

Unbalanced Tree Search.

Data records can be pointed to directly from the highest level node. Data that is required more often can be accessed by a shorter access path through the tree index than other less used data.

Sequence Set Operations.

The bottom level of a balanced tree index is called a sequence set. The entries are normally in collating sequence andso it is quite easy to do a sequential scan of this set to perform a sequential access of the data but should only part of the data require access, the overhead of skipping through the index is far less than skipping through the data. If the nodes of this bottom level index are not physically contiguous then horizontal chains are required between the nodes.

'Look-Aside' Buffers.

This technique is useful in the case of clustering. A particular index node may be in main memory and the range of index values is 'noted' in what are called 'look-aside' buffers so that if an index value is required in that range, the entry is already in main memory. Such a buffer can be used at several or all levels of tree index.

Calculated Address Guessing.

A calculated guess at the position of an index entry may be made. Whether or not this is possible is dependent on the application using the index.

Algorithm Indexing.

In this method the index block is located by algorithm.

Hash Indexing.

The hashing technique is used to locate the index block rather than in pure hashing when the data record is located. This method is not as fast as pure hashing but it has 2 main advantages:-

(1) The data records can be placed in any sequence.
(2) Overflow and the problems of handling overflow within data records are removed to the index.

2.5.11. Index Searching using Key Truncation.

Keys may be very long, drastically reducing the number of keys that can be accessed in one direct access transfer. A method commonly used in directories is to truncate the key and to use the first 'n' characters that have been retained, as a range which points to the next level of index. At the bottom level of index the truncated key can be used to uniquely point to the data.

Tree with Fixed Length Key Truncation.

Figure 2.21. describes such a 3-level tree with the first level in main storage and using a truncation to four characters. There are two other fields in each entry and they are used in two different ways dependent on the type of entry.

If an entry points to a lower level index entry the first of fields points to the index address, in this example a track number. The second field is not used. In the bottom level index the first field points to the address of the data at the head of the list while the second field gives the list length.

In this example the tree is unbalanced because at the second level the 'BABB' pointer is direct to the data list.

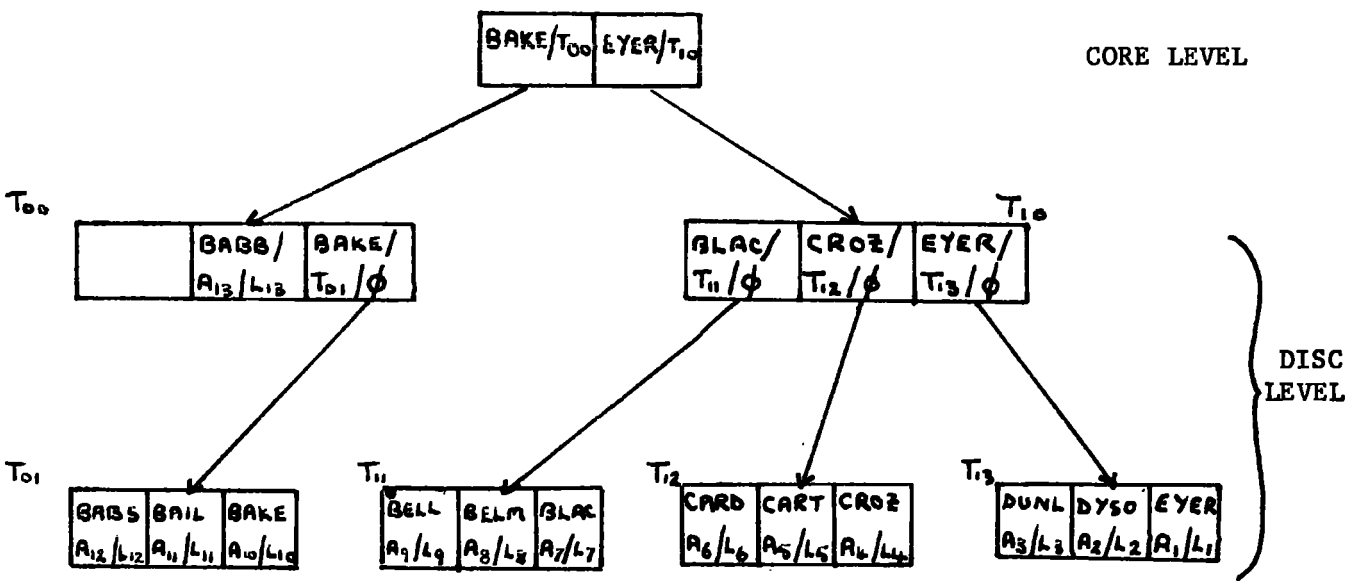


FIG. 2.21. TREE WITH FIXED LENGTH KEY TRUNCATION.

Tree with Variable Length Key Truncation.

Instead of having a fixed length truncation the minimum number of characters are used that uniquely identify the truncated key from any other. One major problem is that the storage required will be variable so that the number of entries per track will also be variable. It is also possible that the decoding will not be unambiguous. Should the key be in the directory then it will be uniquely decoded but if the key is not in the directory, then without comparing the key required with the full key in the directory, decoding may not be unambiguous.

2.5.12. Data Compaction.

Elimination of Redundancy.

One of the basic characteristics of data base management systems is to store a data item only once. This is in itself a logical compaction of data.

Conversion from Human to Compact Notation.

The best example of this is the compaction of dates e.g. a date could require 9 characters to describe it in Human terms - 12 NOV 1976 but this can be compacted into 2 E.B.C.D.I.C. characters as follows:-

YEAR - 7 bits.
MONTH - 4 bits.
DAY - 5 bits.

Suppression of Repeated Characters.

In a string of characters there are often several space characters together. After the first space, an indicator is set to show that the preceding character is repeated. The indicator is followed by a bit string, the numeric value of which gives the number of times the character is to be repeated.

Presence and Absence of Fields.

Variable length records may consist of several data items which may or may not be present. To avoid using space to describe these absent data items a bit map can be used, preceding these data items. A zero bit indicates the items absence and a 1-bit indicates the items presence.

English Text Substitution.

The most common phrases can be coded, if for instance most of the phrases belong to a group of say 200 this group could be coded into a numeric string of bits.

Compression of Sorted Data.

The first characters of sorted data are often repeated. These repeated characters can be coded.

General Purpose Encoding.

Different encoding schemes are application dependent, for instance in section 2.5.11 the directory decoding by truncation of the key is an example of key compression. As a general method, characters can be coded into a variable length list of bits so that the most common characters are encoded into a minimum number of bits. If in a particular application certain characters appear very regularly the saving can be great averaging out to only 3 bits for each character.

Implementation can be by software, microprogramming or special hardware. The more the implementation is oriented to hardware the faster it will be. The greatest benefit is derived if the data is to be transmitted over long and expensive transmission lines.

2.5.13. Relations.

A relation is the viewing of data as a table with the columns (or field types) called domains and the rows equivalent to records called tuples. The basic characteristic of a relation is that there are no repeating groups within the tuple. The characteristics of relations and the part they play in data base management will be described later.

2.5.14. Binary Relations.

Binary relations are a particular type of relation, they only have two domains (columns). This is the most simple of structures and even the most complicated plex structures can be described in this way. Such a structure need only be implemented as the logical schema structure and need not be implemented as the storage structure. The main problem with the binary relation is that the simplicity is gained by a large degree of redundancy but at the logical level this is not necessarily reflected in

the storage structure. The simplicity permits such relations to be searched quickly.

Relations themselves are broken down into binary relations by associating every domain with its identifying domain. This can be almost a 100% increase in logical redundancy.

The binary relation like the inverted list is best suited to interactive query systems.

The advantages are:-

(1) Each relation list is as compact as possible, consequently a large amount of data can be transferred into main store for searching. The searching can be very fast.

(2) The data base has a uniform structure, only the mechanism for manipulating binary relations is required.

(3) Complete flexibility is provided, data items can be easily inserted without change to what is already there.

The disadvantages:-

(1) More storage is required although extensive compression techniques could be used.

(2) To build up the sub-schema record may require searching several binary relations. This is always the problem when building a complex sub-schema structure from a simple schema structure.

2.5.15. Triads.

A binary relation has the form:-

R (x, y)

where R - is the relation

x, y - are the two data items

y is related to x by a relationship R

The grouping of these three components is called a TRIAD.

The following relation 'DEPARTMENT' can be split into 4 triads.

DEPARTMENT (DEPT-#, DEPT-NAME, REPORTS-TO, MANAGER, BUDGET)

(1) DEPT-NAME (DEPT-#, NAME)
(2) DEPT-REPORTS-TO (DEPT-#, EMPLOYEE-#)
(3) DEPT-MANAGER (DEPT-#, EMPLOYEE-#)
(4) DEPT-BUDGET (DEPT-#, BUDGET)

Triads must also be able to describe 'nested' relationships. In the following relationship the 'QUANTITY' data item is related to a 'MONTH', a 'BRANCH OFFICE', an 'ITEM'.

SALES (ITEM-#, BRANCH-OFFICE-#, MONTH, QUANTITY)

The triad schema becomes:-

TRIAD-#: SALES (ITEM-#, TRIAD-#)
TRIAD-#: SALES-BO (BRANCH-OFFICE-#, TRIAD-#)
TRIAD-#: SALES-MONTH (MONTH, QUANTITY)

As an instance of the schema this becomes:-

103 : 1 (7182,902)
902 : 2 (7,670)
670 : 3 (9,2001)

and the triad names are stored separately:-

1001 : 4 (1, SALES)
1002 : 4 (2, SALES-BO)
1003 : 4 (3, SALES-MONTH)
1004 : 4 (4, TRIAD-NAME)

All relationships in a data base can be described in terms of triads.

Triad queries can take any of the following forms:-

R ?? Find all x and y values of triads with an R value.
R x ? Find y values of triads with an R and x pair of values.
R x y Test for the existence of triads with R, x and y values.
Create a triad.
Delete a triad.

2.5.16. Summary of Physical Organisation.

The section started from the basic access techniques showing the different ways of addressing data, whether by hardware address, relative address or record identifier. One particular data field is the record identifier or primary key. The record can have several data elements which are used as paths of access to the record, they are called secondary keys. This was followed by a summary description of some of the search techniques. One such technique is the hashing technique and several hashing algorithms were described. The search path can be pre-defined by 'chaining' records together by means of pointers. There are methods of limiting the size of chains to suite the particular physical device characteristics. It is possible to describe the logical tree and plex structures in a physical representation. The inverted list is a means of separating the relationships between data and the records themselves. There are many ways of searching such index tables and to make the searching more efficient the key values which can be lengthy are truncated to enable as many index entries as possible to be stored in the index block. Data can be stored in a compact form. The different compaction techniques available can be generalised or application dependent.

A particular way to organise data without recourse to indexes or pointers is to use relations. A two element form of relations is called binary relations which is the most generalised way of organising data. The two data elements and the relationship that exists between them is formally described as a triad.

2.6. DATA STRUCTURE DIAGRAMS.

In section 2.2.3. when data relationships were described, any two data elements can be related together and this structure could be described diagramatically by enclosing each data element in a box and connecting each box with an arrow. Such a diagram is a data structure diagram which is sometimes called a'Bachman Diagram' after the documentation technique was first used by Charles W. Bachman in the early 1960's.

The great advantage of such a technique is that it is not limited to describing the organisation's data relationships but any entity or attribute in the fullest and widest meaning of those words. The diagrams can be used to describe the physical data structures and even the different storage structures to be used. It is a method of describing how the different

structural concepts relate together, which permits the different concepts in this chapter to be tied together. Appendix 1 gives example diagrams showing the way data structuring has evolved.

2.7. REFERENCES.

1. "Concepts for Data Base Management Systems"
R. Durchholz and G. Richter.

3. DATA BASE MANAGEMENT SYSTEMS.

3.1. INTRODUCTION.

In the previous chapter the data and its different characteristics were investigated. In this chapter the characteristics of how the data is efficiently and correctly handled is considered.

ACCESS ROUTINES must be provided to access data by different methods dependent on differing access requirements.

Central control software is required to perform this efficient and correct handling of data. This central management software is often referred to and shall be during the remainder of this thesis as the DATA BASE MANAGER.

One of the most important and influential people to be concerned with data management is the person with the title DATA BASE ADMINISTRATOR. The Data Base Management System (henceforth referred to as D.B.M.S.) should provide the administrator with several utility sub-systems to assist in the controlling of the data resource.

Associated with the storage and accessing of the data base are several languages. The number and function of such languages vary from D.B.M.S. to D.B.M.S. The different functions performed are:-

(1) To describe the way the data is stored.
(2) To describe the data and data relationships in the schema.
(3) To describe the data and data relationships in the sub-schema.
(4) To permit the requesting and manipulation of data.

Associated with the manipulation of the data base, the D.B.M.S. packages can be separated into 2 separate groups called:-

(1) Self-contained D.B.M.S.
(2) Host D.B.M.S.

The self-contained system provides a complete language to access, manipulate and process the data base data while the host system provides methods of data base access within existing high level application languages

by the addition of new language verbs or the use of interface subroutines which are CALLed.

There are commonly held views on the advantages and disadvantages of D.B.M.S.

The advantages:-

(1) Minimises redundant data (storage of the same data more than once) which in turn reduces the cost of storage.

(2) Sorting runs should be eliminated.

(3) Program costs are reduced because the complexity of programs using the D.B.M.S. are simpler than their non - D.B.M.S. counterparts, because much of the data handling processing is not required.

(4) Control of access to/contents of the data base as provided by the D.B.M.S. brings with it special advantages not realised without the data base capability.

3.2. DATA ACCESSING.

The way data is accessed can be a good example of how certain aspects covered in the previous chapter tie together. Figure 3.1. describes diagramatically the five facets of data detailed in section 2.2. and also the files and procedures necessary to access such a structure.

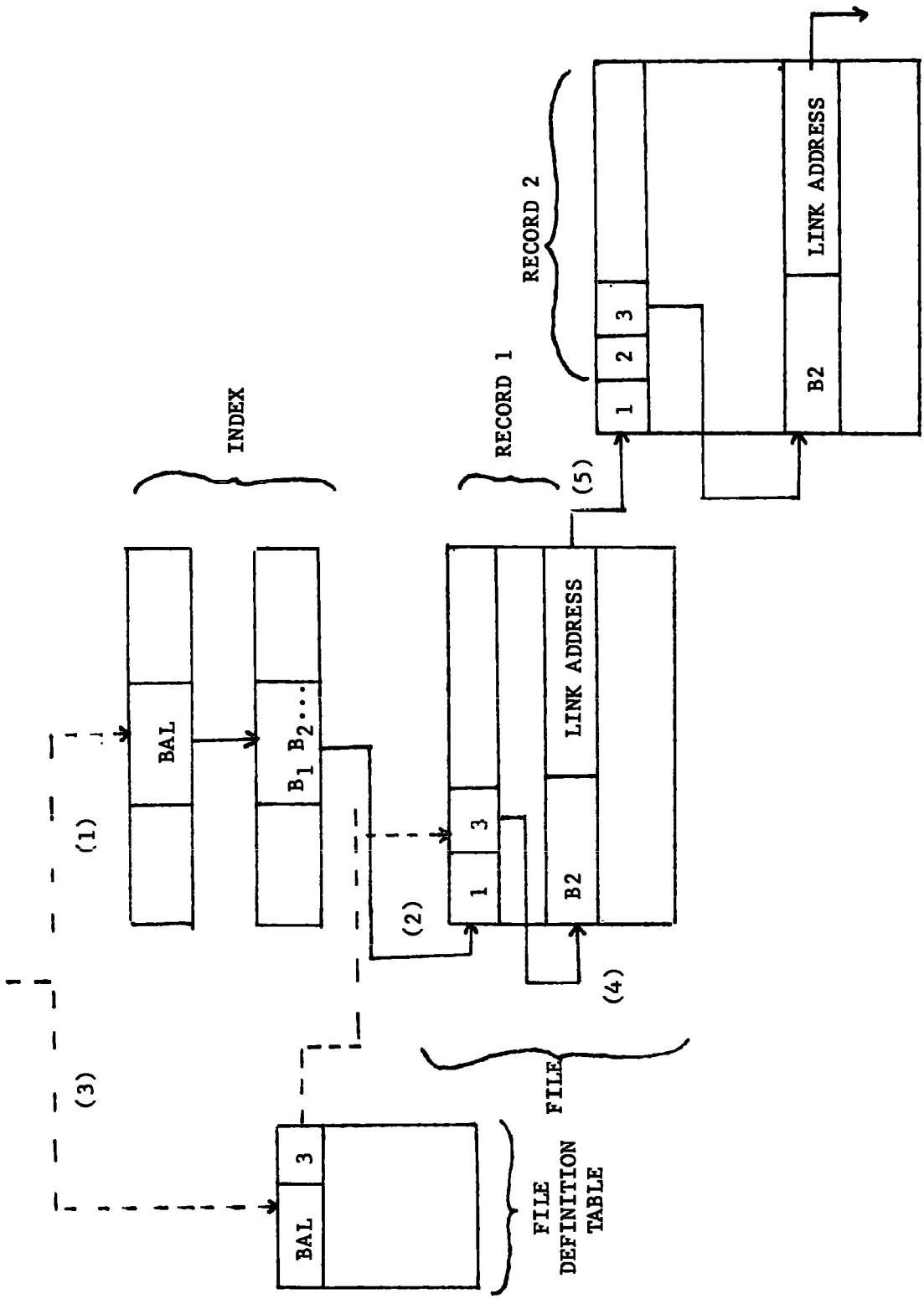

INDEX
RECORD 1
RECORD 2
FILE
FILE DEFINITION TABLE
BAL
B_1 B_2...
LINK ADDRESS
B2
1
2
3
(1)
(2)
(3)
(4)
(5)

FIG. 3.1.

There are 4 types of files that may be required in a D.B.M.S. access:-

(1) File Definition File - defines the record structure in the sub-schema and schema. This file is more often called a DATA DICTIONARY. It should catalogue the whole description of data and should be the basis of all information concerning the data base for users, programmers, auditors, managers and data base administration staff.

(2) DIRECTORY - this file is often confused with the data dictionary, however the functions of the two files are entirely separate. The purpose of the file is to translate the request for data into a set of record addresses within the data base. The directory drives the D.B.M.S. in its access to the data base. It contains the access control information and mapping information.

(3) Inverted List File - The directory may point to an inverted list file rather than a normal data record. This will be the case if an access is made as a result of the enquiry 'give me the identifiers of all entities x which have an attribute y value of z'. If there is a requirement for this type of enquiry against a data base the inverted list becomes an integral part of such a data base.

(4) The Data File - the data records may contain pointers to other data records as well as the basic data itself.

3.2.1. An Example of the Facets of Data.

Figure 3.1. shows 3 different record types:-

(1) The file definition table.
(2) The index record.
(3) The data record.

The illustration is of an access for a data item 'BAL' with a current value represented as 'B2'. The data 'BAL/B2' is associated with 2 data records.

The access is in 5 steps:-

(1) The index is accessed for a data item 'BAL' with a value of 'B2'. The index has 2 levels. The first translates 'BAL' and points to a second level which contains a list of possible values for 'BAL', symbolically represented as B1, B2 etc.

(2) The value index is translated to give the address of the head-of-list address of the list where 'the value of BAL is B2'.

(3) The file definition table must be consulted to unpack the contents of the record.

(4) The record may have a table of contents so that the value of 'BAL' is accessed by taking the third entry in the table and using the value as an offset address to find the location of 'B2'. The file definition table is identifying 'BAL' as the third data element in the record.

(5) Associated with this data elements there is an address field which points to the next record in this 'BAL = B2' list.

Looking at this example, the facets of the data accessed are:-

The Semantics.

The extent or definition of the data element is 'BAL/B2', where 'BAL' is the name of the data element and 'B2' its symbolic representation. The intent of 'BAL/B2' is 'an amount of money, B2, owed by a customer, represented in the system by a record'.

The Syntax.

There are several syntactic relations of the data element that are apparent:-

(a) The amount of money represents a set of values 'B1, B2...' and this set is expressed as a two-level index. 'BAL' points to the values at the second level.

(b) The values 'B1, B2' have relations to one another, such as >, <, =.

(c) Several records may share the same balance value this is identified by the list of records each containing a value for 'BAL' of 'B2'.

(d) Time sequence is implied by the order of the records on the file.

(e) The data element referred to as a 'balance' may be a component of a larger aggregate such as 'accounts receivable'.

The Representation.

There are 2 types of representation. The explicit representation is 'BAL' appearing in the index and in the table which has a value 'B2' which is represented in example form as '195.50' in both the index and the record. The format of 'BAL' is 'three alphabetic characters' and the format of the value as 'a fixed point number with two places of decimal precision and a total of up to eight significant digits'. The implicit representation is, for example, equivalent to the sum of all record balances.

The Hierarchy of Data Aggregates.

This is illustrated by the fact that all data starts from the most primitive symbols of the computer, binary 0 and 1.

The Functional Aspect of Data.

The user-derived data is 'BAL' and '195.50' whereas the system-derived data is index pointers and table entries.

3.2.2. On-Line Updating.

After looking at the physical structure of data in the previous chapter, it is worth looking at the impact of performing on-line updating on these structures and the possible effects of failure while such updating is taking place.

There are 5 categories of updating:-

(1) Whole record addition.
(2) Whole record deletion.
(3) Deletion of Keys.
(4) Addition/deletion/modification of non-keyed data.
(5) Addition of Keys.

There are 2 basic problems:-

(1) In every category of update except possibly (4), the directory requires updating and/or pointers require amending.

(2) When a record is updated it may increase in size, necessitating the record to be relocated. This is possible in categories (4) and (5) above.

3.2.3. Updating of a Multi-list File.

Any record may occur in several different lists although it may only be stored once. This infers:-

(1) When the record is deleted it must be clear to the updating process whether the record is to be physically deleted or simply to mark the record deleted with a delete bit, that is, the record is left on file until a reorganisation process removes the record from the file.

(2) Each of the record's secondary keys could be accessed through a separate list. A delete bit is required for each key to indicate the record's continued presence in the particular list.

Whole Record Addition.

The following steps are required to add a new record to a multi-list structure:-

(1) The record is edited prior to being written to file.

(2) The record's address is obtained by one of several methods.

The following steps are performed iteratively taking each of the record's secondary keys in turn and inserting a pointer to this record for each of the list of which this record is a part.

(3) The particular secondary key (key x) is decoded in the directory.

(4) The address in the directory obtained by this decoding is the head-of-the-list address which is transferred to the link address field of the new record.

(5) The address of the new record becomes the new head-of-list address for this key.

(6) The list length is incremented by 1.

(7) After steps (3) to (6) have been repeated for each secondary key the new record is inserted at the correct address.

Whole Record Deletion.

The delete bit is set in the record. If the list length is the physical list length this is the end of the deletion process, but if the list length kept is a logical length i.e. the number of records to be retrieved in an enquiry, then each key in the record must be decoded to find all the lists containing this record and decrement the list length of each.

The control fields required in each record on a list are shown in figure 3.2.

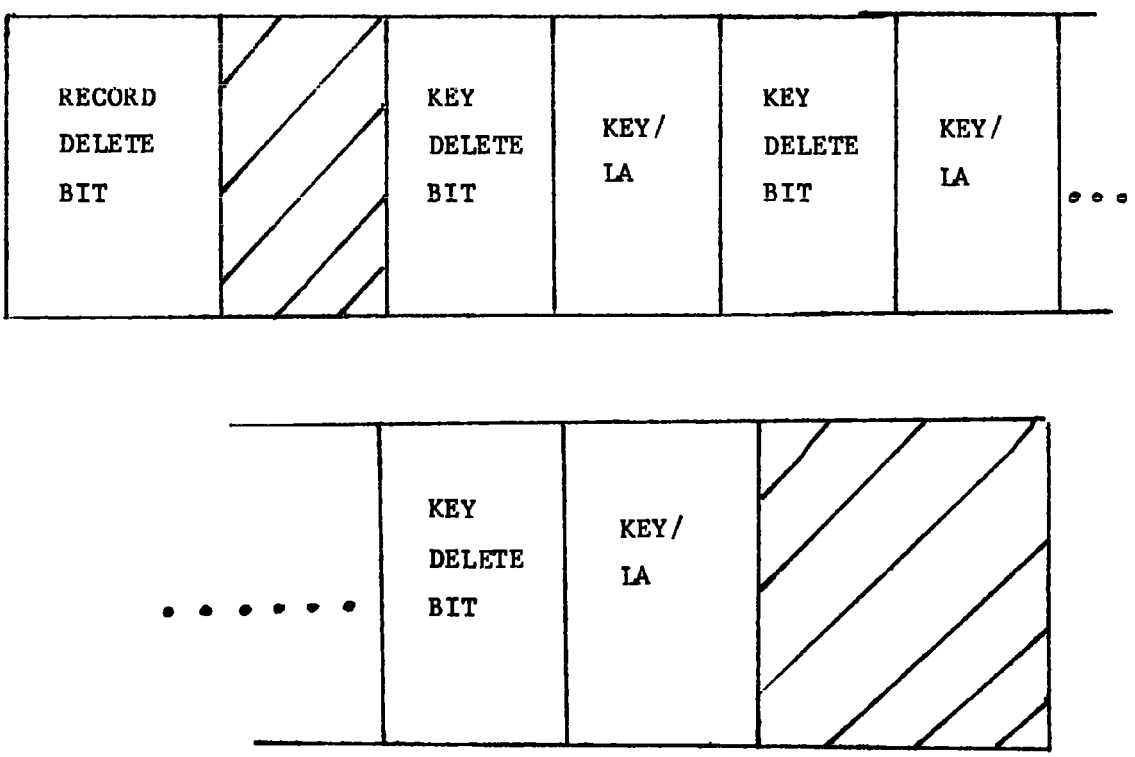


FIGURE 3.2. MULTI-LIST RECORD UPDATE CONTROLS.

An alternative to this deletion technique is to break the list, and in the example of deleting record N, to set the pointer to record N + 1 in record N - 1. There is little disadvantage if record N is accessed when deleted, particularly if a cellular multi-list structure is used because the likelihood is that the other records in the list will be in the same cell as the deleted record. The disadvantage of this method is that there will be recovery problems if there is a failure in the middle of the pointer update, as shall be seen in the section on recovery. There would also be a big overhead because this pointer manipulation would need to be done in every list that contained the deleted record.

Key Deletion.

Exactly the same procedure is required to delete keys although this time the key delete bit is set. The list length for the lists using the key can be decremented if required.

Addition/Deletion/Modification of Non-key Data.

If the record is shortened the data within the record need only be packed. If the modified data has increased in size, but the increase does not cause overflow, the physical block is written back. If overflow is caused, the record is written in modified form in a different physical block as though it were a record addition. The record that was read has its delete bit set and it will be physically deleted in the periodic reorganisation.

Addition of Key Data.

The following stages are used to perform this function:-

(1) The key(s) are added to the record.

(2) Each key is decoded in the directory.

(3) The head-of-list of this key is transferred from the directory to the link address field of the key in the modified record.

(4) The address of this record becomes the head-of-list address of the key in the directory.

(5) The list length for this particular key is incremented by one.

(6) Steps (3) to (5) are repeated for all keys to be added.

(7) The modified record is restored according to a non-key modification procedure as described above.

3.2.4. Updating an Inverted List File.

Whole Record Addition.

The following steps are followed:-

(1) Set up record contents ready to be put on to the file.

(2) Assign an address to the record.

Steps (3) to (5) are repeated for each key within the record.

(3) The key is decoded to give the entry for the variable length inverted list.

(4) The address of this new record is inserted in the correct sequence in this list.

(5) The list length is incremented.

(6) The new record is stored in the correct location on the disc.

Whole Record Deletion

(1) The delete bit is set in the record.

(2) All keys are decoded and the inverted lists on which these keys appear have their list lengths decremented by 1.

This is a logical deletion. In addition to the above steps, the particular addresses held within the lists for this record can be removed.

Deletion and Addition of Individual Keys.

These updating procedures are a subset operation of the whole record deletion and addition procedure.

If a cellular partitioning physical organisation is employed there is no need to update the directories unless a record is transferred into or out of a cell.

3.2.5. Space Maintenance.

In the update process a large amount of unused space is left through the relocation of records or the setting of record delete bits. The need is to make this space useable again.

There are two types of space maintenance:-

(1) The re-claiming of space off-line to the update process. This is the easiest method, provided time can be found periodically to do the reorganisation process. The cost of file regeneration must be balanced against the optimum time interval between reorganisation so that the reorganisation process does not become prohibitive.

(2) The reorganisation process may be performed in background to the updating process.

The space maintenance can be either performed logically or physically. The logical method puts the unused block at the head of a list of unused blocks so that the free space is chained together. The physical method collects the unused space into physically separate areas. The latter method is more efficient for real time updating. There is a danger with logical maintenance in that the list of available space itself may require maintaining. The maintenance process must be able to be suspended to permit records to be moved.

3.2.6. Random File Update Integrity.

The basic problem in any updating process is that there will usually be more than one write operation in the single update process, so that there is an integrity problem should a failure occur between the write operations. The key to providing automatic file integrity recovery is provided by:-

(1) The order in which operations are performed.

(2) An algorithm that permits a corrupt file to be detected and either corrected or restored via back up.

It is worth looking at the updating integrity problems associated with three different files.

Single Keyed File - what happens in record addition?
what happens in record deletion?

Addition - Step 1 - the block where the record is to be placed is read and space is allocated.

Addition - Step 2 - the record is added to the block which is written back.

Addition - Step 3 - the index is updated.

There is no problem at any stage of the process unless there is more than one index required to be updated and therefore more than 1 index write is required. This problem can be overcome by changing the sequence of steps to 1, 3 and 2. In the recovery process the index is read and if the key is not present the entire update process is repeated. If the key is present the data record is read. If the record does not contain the right key then steps 1 and 2 must be repeated.

Deletion - Step 1 - Read block containing data record.

Deletion - Step 2 - Update Index.

Deletion - Step 3 - The record is deleted, the space is released and the block is written back.

The only possible failure consequence is that after step 2, space will be reserved for the non-index record.

If a record is moved this is really a concatenation of the deletion followed by the addition process.

Chained List Structure.

The steps to add a record to a chained list have already been described. The failure to successfully complete the setting up of all the keys in the index will lead to a loss of integrity which cannot be recovered by using the data within the file, other recovery data must be available. This infers a loss of file integrity.

In the case of a key being updated in a record in a chained list structure, the record is logically deleted from the old list to be added to the new list.

The steps involved are as follows:-

(1) Read data record.

(2) Set the key delete flag.

(3) Decode and add new key to the new index.

(4) Allocate space (record may grow) and write data record.

(5) Write the index.

If a failure occurs after step 4 there is a loss of data integrity because the record will have been logically removed from the old list but not yet added to the new list. This failure can be recovered from within the file since the record is only marked as deleted from the old list but is nevertheless still chained in that list.

Inverted List Structure.

There are 3 basic steps involved in adding a record to an inverted list structure:-

(1) Read current data block.

(2) Add record, allocate space and write data block.

(3) Decode index, add record address (or key value) and write list.

If failure should occur during the second phase, space will have been reserved for a non-indexed record. If failure occurs during the third phase the data integrity will be lost, with the data record on the file without the correct list entry. If there are n keys in the record, then there will be n lists to be updated. Some but not all of the lists may have been updated. Provided it is known that a crash has occurred the state is recoverable by repeating the list processing ensuring that the same key is not put on the same list twice.

There are 2 methods to delete a record from an inverted list structure. The first method deletes a record both logically and physically while the second simply indicates the record is deleted (logical deletion).

In both cases the data record block is read. In the physical deletion method, for each of the keys in the record, the index is decoded and the index key is removed. Finally the data record is deleted, and the block is written back. The only possible loss of data integrity is in the removal of the index entries. The problem is identical to the insertion of an inverted list record. The second method of deletion overcomes this problem by simply setting the record delete flag in the record and writing the block containing the record.

3.3. D.B.M.S. MECHANICS.

Having described an example of data accessing, it is worth looking at how a D.B.M.S. might work and how such a D.B.M.S. would interface with operating system, secondary storage and the applications using the D.B.M.S.

Figure 3.3. gives a diagramatic representation of what could be the paths through a D.B.M.S.

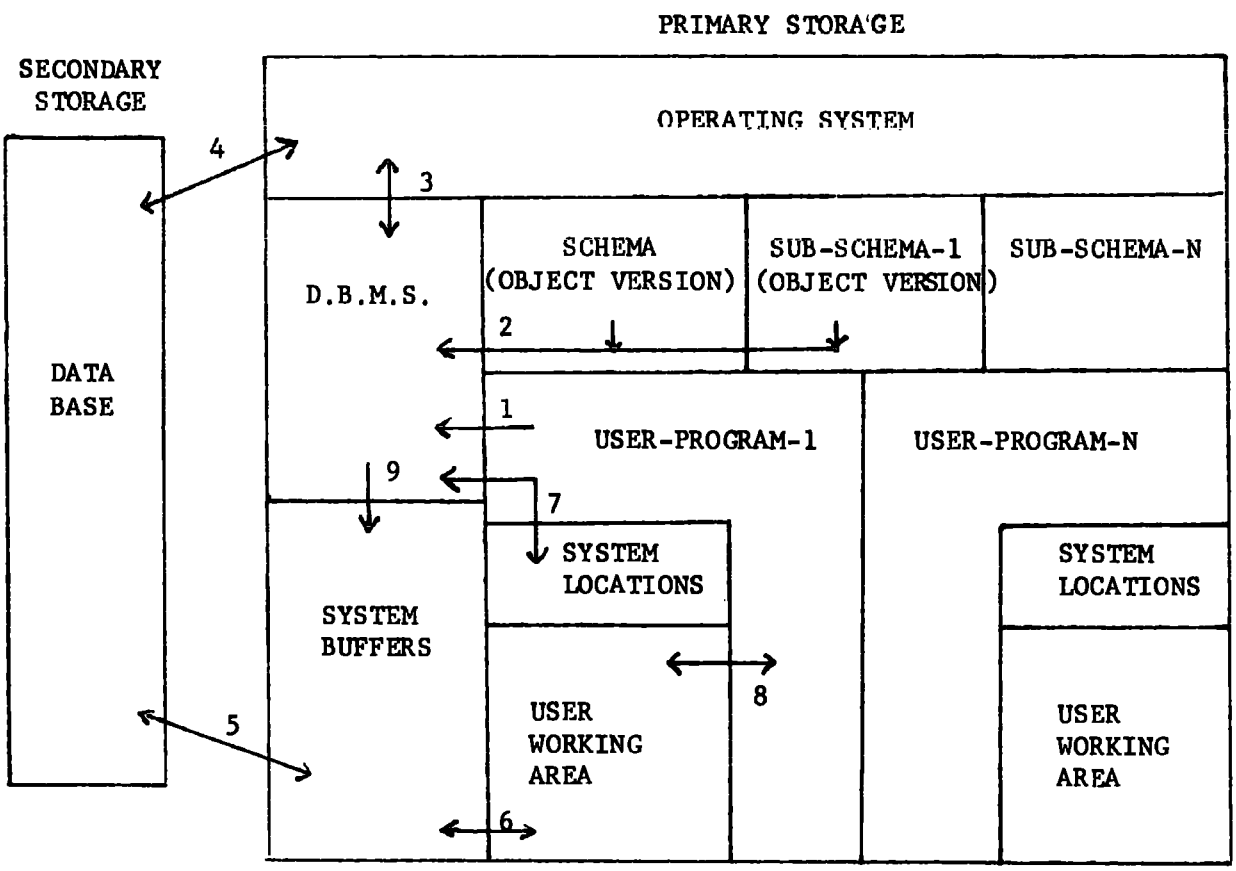


FIGURE 3.3. CONCEPTUAL DATA BASE MANAGEMENT SYSTEM.

The steps involved in requesting data are as follows:-

Step 1 - A user program makes a call to the D.B.M.S. for data. The request is made by data manipulation language (D.M.L.) commands.

Step 2 - The D.B.M.S. analyses the call and supplements the arguments in the call with schema and sub-schema information. The sub-schema may consist of only the differences from the schema, it may not be a complete subset of the schema.

Step 3 - On the basis of the information from the schema and sub-schema and from the call itself, the D.B.M.S. makes the physical input - output request through the operating system.

Step 4 - The operating system interacts with secondary storage.

Step 5 - The operating system transfers data between secondary storage and the system buffers.

Step 6 - The data is transferred from the buffers to the user work area by the D.B.M.S. Any change between the picture of the data the user sees and the picture of the data on the storage device, is made during this transfer.

Step 7 - The D.B.M.S. provides status information to the calling program indicating the result of the call.

Step 8 - The host language manipulates the data in the user work area.

Step 9 - The D.B.M.S. must service the system buffers which are shared by all the programs.

This example is of a conceptual D.B.M.S. as devised by a CODASYL committee. The work of the CODASYL committee will be described in a later chapter.

3.4. D.B.M.S. LANGUAGES.

The following list of languages are representative of the language functions required in a D.B.M.S. environment.

3.4.1. The Physical Data Description Language.

This type of language describes how data is placed on the storage media. It indicates how buffering, paging and overflow are handled and controlled. The language specifies the addressing and searching schemes to be used when accessing the data. This facility may not be provided by a language in some of today's D.B.M.S., instead a utility program might be used. Nevertheless no matter how implemented a facility of this nature is required.

The CODASYL committee refer to such a language as a Device Media Control Language.

3.4.2. The Schema Description Language.

The view represented by the schema can be quite different from the way the data is stored. The schema description language describes the global logical records and describes how these records are derived from the physical data.

The language could be:-

(1) A D.B.M.S. facility provided to describe the data but independent of all host languages which the D.B.M.S. supports.

(2) A completely independent data description language that can be employed by a future D.B.M.S.

The current implementations are almost exclusively of the first type with the data description language as a part of the whole D.B.M.S. product, although the I.B.M. data description language DL/1 can be used alone or with other I.B.M. data handling products.

The language does not only describe how logical records are derived but may give the extra facilities:-

(1) To define logging requirements when updating data base files.

(2) To define access authority mechanisms.

(3) To define data validation procedures for any data input to the data base.

(4) To provide a description of any data conversion between the physical and logical records.

(5) To define the creation of virtual fields which are fields not stored physically but which are created by performing a function on one or more stored fields.

3.4.3. The Sub-Schema Description Language.

The application programmer must be able to describe the data he uses in his program. There are three types of sub-schema facility to choose from:-

(1) A programming language extension (e.g. COBOL).

(2) A D.B.M.S. facility.

(3) An independent language.

As has been said in the previous chapter, the sub-schema view is a logical, or subset, view of the global picture in the schema. The sub-schema description language describes the parts of the schema that are of interest to the application program and within this subset it may only describe the data characteristics that are different from the schema description, so that, should certain details of the data view be missing, the D.B.M.S. will default to the details in the schema description.

3.4.4. Programming Languages.

In host D.B.M.S. systems the application programmer will write his program in a conventional programming language whether such a language is a high level language or an assembler.

3.4.5. Data Manipulation Language.

A manipulation language must be provided to permit the application program to give instructions to the D.B.M.S. and to receive and interpret status messages which indicate whether the request has been satisfactorily accomplished. The manipulation language may take 1 of 3 forms:-

(a) As an extension of the application language in which case the language could be independent of any particular D.B.M.S.

(b) The data manipulation language may consist of a set of CALL statements in which case they will be largely independent of the processing language, the only variable being the individual language's syntax.

(c) The language may be a separate sub-language in itself, this may allow the language to be independent of any programming language and any D.B.M.S.

The above 5 languages are the basic language tools in a data base management environment. The work of the CODASYL committee in the data base area is mainly concerned with the definition of 3 of the 5 languages described above.

3.4.6. Data Interrogation Facilities.

An additional set of language/utility facilities are often provided to give ad hoc enquiry on the data base. It is becoming increasingly important that if a D.B.M.S. software package is to be commercially viable it must include such facilities.

The following 3 types of facilities are often provided:-

(1) A data base report generator facility to structure reports with a minimum of effort. The facility may be used only in an off-line mode to the data base.

(2) Enquiry facilities possibly provided in a very rudimentary form using parameter or by possibly using a near-English query language.

(3) The possible use of a full on-line data base interrogation, search and manipulation language which would include the above facilities and others including pre-search statistics and the use of application packages and models to provide more information from the data enquired upon.

3.5. DATA INDEPENDENCE.

The more a process is automated the more difficult that process is to change. A clerical system is relatively easy to change, such alterations are communicated to those operating the system, but in changing a computer system, the effects of the changes must be investigated before making the change and re-testing the part of the computer system that has been affected, often a very long and expensive process. In a world of constant change, it is not difficult for a data processing department to find itself spending more time changing programs than developing new ones.

The data independence concept protects the investment in programs by rendering them less liable to change. This is because a high proportion of programming changes is as a result of changes to the data itself rather than changes to the way the data is processed. Data independence is provided by removing the definition of data in the data base from the program that uses it.

In the conventional approach the definition of data is local to the program, however certain characteristics of the data such as file organisation and access are included in such a definition, with the result that the program will often need to be changed, with for example a change in record length, although the actual data fields which are processed may be unchanged.

The data base approach permits data and data structures/organisation to be changed without changing the program accessing the data base. This of course cannot prevent programs being changed when the data fields which the program processes are changed. This independence of data from the processing unit is provided by the 2-level logical data structure of sub-schema and schema as described in the previous chapter and it is a popularly held view that this is the main benefit of using a sub-schema and schema philosophy.

3.5.1. Sub-schema and Schema Architecture.

The schema is a description of all data in the data base it has 2 parts:-

(1) A picture of the data base describing all data elements in terms of their natural structure and association independent of all considerations concerning physical representation.

(2) The mapping of the picture into physical storage and the specification of access method.

If the data base is reorganised (the picture is changed) and the map between the data base fields/records/files and the physical storage must be altered, so that, despite physical storage changes, the picture of the data base to the programmer/user remains invariant and to quote from reference 2:-

"the ability of the application programs to execute correctly regardless of the way their data is actually represented and accessed in the data base."

The sub-schema gives a picture of the structure of the data as it is to be processed which may be contrary to the data's natural structure as exhibited in the schema. It is a set of logical file definitions. The sub-schema provides the basis for the independence of the application program which uses the sub-schema, while the schema, as a representation of the organisational data structures, may change but which should only precipitate such a change to the sub-schema of the program if the processing within the program is itself affected by the data change. Such a sub-schema may be used in more than one program if the way the data is looked at is the same in more than one such program.

There is no reason why any problem solving high level language should not be provided with an interface to a data base using the sub-schema/schema architecture but such languages should be extended if they are to be adequate in a data base environment, providing:-

- much better (extensive and flexible) record selection procedures;
- much better data structure handling facilities.

There are several major advantages that the sub-schema/schema architecture has over other D.B.M.S. that may not use this approach:-

(1) The individual application programs are interested only in a small portion of the data base. Only a subset of fields and records are of interest and the program should not be provided with access to the records and fields which aren't of interest.

(2) Names given to data elements in the sub-schema which are used by the program may be different to the names of the same data elements in the schema i.e. data mapping is provided.

(3) The logical records in the sub-schema should permit data elements from different schema records to be used to build up the sub-schema logical record. The logical record should be allowed to describe complex hierarchical or network structures which may be totally different from the structure from which the fields were obtained in the schema.

(4) Such a restructuring may be required to permit existing application programs (existing investment) to use a newly developed data base.

(5) The capability to leave a program unaltered when the fields and records not used by the program are changed in the schema.

The leaving of application programs unaltered when the schema is changed is described as LOGICAL DATA INDEPENDENCE. The changing of the way data is stored, indeed the changing of storage hardware itself, while leaving the schema unchanged, and by implication the sub-schema unchanged is called PHYSICAL DATA INDEPENDENCE.

3.5.2. Logical Data Independence.

What are the reasons for changing the global logical data view as described in the schema?

(1) The adding of a new field to the schema record type which is not of interest to any currently existing application.

(2) Deleting a 'field of interest', any application using this field will cease to function without amendment.

(3) The addition of a whole new entity set to the organisation's information model may precipitate the addition and extension of several schema record types. This should leave the sub-schemas, which use data types based on original entities, to remain unaffected.

(4) The moving of a field from one schema record type to another schema record type. This is clearly a combination of (1) and (2) above, which necessitates a change in the mapping between schema and sub-schema.

(5) A one-to-one relationship may be changed to a one-to-n relationship. If an application program assumes a one-to-one relationship it will very likely need re-writing. A method of overcoming this problem for the case where data elements are not referred to independently e.g. as a summation of the data values, is to change the sub-schema to produce a single virtual field with the value of the summation already computed from the individual values.

3.5.3. Application Program Discipline.

The basic flaw in programming discipline which destroys logical data independence is the assumption that a data element (record or field) has a set number of occurrences. Should the program assume that a field has only one occurrence and at a later date several occurrences are introduced and the processing demands that each occurrence should be handled independently then the program will almost certainly have to be rewritten.

3.5.4. Physical Data Independence.

If a D.B.M.S. provides PHYSICAL DATA INDEPENDENCE it provides the ability for application programs to execute correctly regardless of physical representation chosen and method used to access it.

The physical representation may additionally include 'spurious associations' which are relationships introduced into the data base by the physical storage structure over and above the relationships defined in the schema. The 'natural associations' are those that are defined in the schema linking fields to form records. Therefore a 'spurious association'is one which links one or more schema records together. A 'pure representation' consists of only the 'natural associations'. In this sense indices do not exist and key fields have no significance. Such a 'pure representation' then can only be found by serial scan.

This is obviously inadequate for performance reasons resulting in the following compromises to the schema's natural relationships:-

- the addition of indices to speed up the search operations;
- the ability to split schema records into several stored records, permitting the most commonly accessed fields to be stored on the fastest devices;
- the addition of pointer chains to speed up searching;
- the storage of hierarchically related data in physical hierarchical form to speed up the retrieval operation.

If the stored field is considered the smallest stored data unit, then several such fields are grouped into a stored record. Each stored record occurrence has an unique logical address assigned to it by the D.B.M.S. when first created in the data base. By keeping the logical address distinct from the physical address simplifies the reorganisation of the data base because the logical address can be used in the spurious pointer associations between stored records and are unaffected by any data base reorganisation and do not need updating as a result of such a process.

The 'anchor record' is the stored record which contains the identification value corresponding to the schema record occurrence. There must be a map between this schema record and the stored anchor record. The possible differences between schema and stored record are as follows:-

(1) The schema record may be spread over 2 or more stored records.

(2) If the identifier is stored redundantly, the schema record is said to have multiple anchor records (the situation when a record is indexed).

It must be possible to find all stored forms of schema field occurrences. Such schema field occurrences may be represented by the following methods:-

(1) Physical Position - the field occurrences are stored as part of the anchor record itself.

(2) Forward Pointer - there is a pointer from the anchor record to the record containing the field.

(3) Backward Pointer - there is a pointer from the record containing the field to the anchor record. In this case the field is not accessible from the anchor record.

(4) The field may be stored redundantly in both the anchor and remote record. Care must be taken to ensure consistency.

With physical data independence the following points should be in evidence to the user:-

(1) The user is unaware of what device is used, this capability is often provided by the operating system anyway.

(2) The user is unaware of a data unit's physical address.

(3) There is no concern for space utilisation and how space is used.

(4) The user is unaware of the physical representation of data, the best representation method is often dependent on value or frequency of usage.

3.5.5. Degree of Data Independence.

It is very difficult to have a program fully dependent on the data accessed, likewise it is virtually impossible and very inefficient to have 100 per cent data independence. The independence relationship between a program and data is a range and the position within that range is known as the 'degree of independence'. This degree is only a roughly calculated and very relative value within the range. It can in some way be evaluated by looking at the following 5 aspects of comparing the logical programming definition with the physical storage definition:-

(1) Equivalent data elements may differ in usage i.e. in terms of size, scale, precision and alignment. Editing symbols may be provided to convert a physical data element to its equivalent logical data element. This is in part equivalent to the MOVE verb in COBOL when a data field with a particular 'picture' is moved to a different data field with a different 'picture'. Many of the current data base management systems do not operate at the data item level and so do not provide such a facility.

(2) Equivalent record structures may differ such that the logical record is a subset of the physical record. This enables new fields to be added to existing physical records without altering the programs which do not access these new fields.

(3) A logical record may be constructed by extracting data elements from a number of physical records. It is not important which physical records contain the data elements required i.e. data elements can be moved from one record to another without affecting the program.

(4) The facility may be provided whereby data elements of a logical record do not exist physically but are constructed from other data elements. Often such elements are referred to as 'virtual elements'.

(5) The logical and physical files can differ in organisation. The program may have to make allowances for access method used by the management system. Many D.B.M.S. permit this independence of program from access method by restricting the logical organisation.

3.5.6. Symbol Resolution (Binding).

This describes the extent and time at which the logical organisation is bound to the physical storage organisation. The time and extent of the 'binding' can affect the degree of data independence obtained.

The process of symbol resolution or as it is more popularly known as binding, has a number of fundamental aspects:-

(1) Via a succession of mappings, the associations between an application program's unique names and descriptions of data elements and the unique names and descriptions of the corresponding stored data fields.

(2) The point and extent at which the resolution occurs.

(3) The resolution process has parameters of extent and time.

The symbol resolution involves not only the association between name and address but also access path definition, data representation and the data elements necessary to completely and uniquely satisfy the application programs data requirements from the stored data. Resolution can occur at various points in the process and to various extents at each point in the process. In materialising virtual data elements it may be best to resolve bi-directionally i.e. by resolving in part from the storage structure and also by resolving in part from the application program.

The different possible timings for binding are as follows:-

(1) On execution of the data base access request in the user program. The method of obtaining the logical record from the physical data definition is determine at execution time.

(2) On Opening the Logical File.

This is a similar method to (1) but the resolution occurs only once per logical file per run making this method more efficient than method (1).

(3) On Loading of the Program.

This method is potentially the most efficient with the data base access command compiled into the object code. The access commands are replaced by routines required to provide the logical record construction from the physical data as it exists at that time. At load time the program's logical data requirements must be ensured to be valid.

(4) On Compilation of the Application Program.

This is the simplest of the methods but with the disadvantage that programs are required to be re-compiled when the data base structure changes. The problem can be reduced if the user is informed about which programs require re-compiling for any data base structure change.

It is possible to bind some data elements at compile time, some at execution time and some at design time. Such partial binding must be done consistently. The decision of where and when to do the resolution can be a major factor in the implementation of data base systems.

Present-day the binding normally takes place at compile time, but eventually there will be a migration to D.B.M.S. which provide a range of binding options, consequently the ability to selectively resolve data elements and conversely to disassociate certain elements. This means the technology is moving towards interpretive data access while the data base is restructured.

The only way of maintaining data independence in present-day installations is to either re-compile programs or dump/restore the data base or both. This method becomes unacceptable for large installations or indeed for any installation in the long term as hardware costs decrease and the cost of people increases, and so the need for more data independence will become apparent.

3.5.7. Normalisation.

The concept of relations described in section 2.5.13. is simply to regard data as though it is in tabular form.

Normalisation is a step-by-step process which converts relations into simpler and more regular structures. The conversion frees the relationships from what are called undesirable associations. Normalisation is often performed on the schema structure to remove the undesirable associations. This clean structure can become the start point of all implemented data structures because it is independent of the storage structures which aid accessing efficiency, and independent of the user data structures which provide the user with the data in the way he wishes to have it structured. This clean and uniform schema will aid data independence because:-

- spurious interactions and undesirable side effects are removed so that:-

- fewer normalised relations will be affected should changes occur in applications or data requirements.

As an example of the normalisation process, a business has an unnormalised relation describing 'orders received for a product':-

BUSINESS RELATION.

Product Number	Product Description	Quantity On Hand	Order Number	Export Order	Customer Number	Customer Priority	Quantity Ordered
B (PN,	PD,	QOH,	OR(ON,	X,	CN,	CP,	Q))
142	NUTS	601	23G	X	4629	1	144
			495	N	3751	8	200
830	BOLTS	95	27K	N	2007	6	12
			23G	X	4629	1	10
			98Y	N	3751	8	36

FIGURE 3.4. UNNORMALISED RELATION.

In this example the data elements (domains) in a relation are grouped within parentheses. The underlined domain is the identifier domain for the particular relation and so in this example there is an 'order details' relation within a 'business' relation i.e. a repeating group.

The first step in normalisation is to simplify the nested relation into two or more relations which will be in FIRST NORMAL FORM.
This gives:-

PRODUCTS

P (PN, PD, QOH)

142	NUTS	375
830	BOLTS	64

ORDER DETAILS

OD (PN, ON, X, CN, CP, Q)

142	23G	X	4629	1	144
142	49S	N	3751	8	200
830	27K	N	2007	6	12
830	23G	X	4629	1	10
830	98Y	N	3751	8	36

FIGURE 3.5. FIRST NORMAL FORM.

In the second step the relations are further broken down so that all non-primary key data is fully dependent on the primary key.
In the above example the 'order details' primary key is a concatenation of 'part number' and 'order number'.

Data concerning a new customer cannot be recorded until the customer places an order, while, if a particular product is deleted, all data concerning the customer who has ordered that product will be lost.

The second step in the normalisation process removes the functional dependencies of the customer details on the 'part number'. This is achieved by splitting the 'order details' relation into 'product orders' and 'customer orders' relations. This gives:-

PRODUCTS			PRODUCT ORDERS			CUSTOMER ORDERS			
P (PN,	PD,	QOH)	PO (PN,	ON,	Q)	CO (ON,	X,	CN,	CP)
142	NUTS	601	142	23G	144	23G	X	4629	1
830	BOLTS	95	142	49S	200	49S	N	3751	8
			830	27K	12	27K	N	2007	6
			830	23G	10	98Y	N	3751	8
			830	98Y	36				

FIGURE 3.6. RELATIONS IN SECOND NORMAL FORM.

The third and final step is to remove TRANSITIVE DEPENDENCIES from any relation. Transitive dependencies are of the form attribute C is dependent on attribute B is dependent on attribute A, then to assume that attribute C is dependent on attribute A, is such a dependency. In this example the dependency of customer priority on 'customer number' which is dependent on 'order number' means that 'customer priority' is transitively dependent on 'order number'. This gives:-

PRODUCTS			PRODUCT ORDERS			ORDERS		CUSTOMER		
P (PN,	PD,	QOH)	PO(PN,	ON,	Q)	O (ON,	CN)	C(CN,	CP,	X)
142	NUTS	601	142	23G	144	23G	4629	4629	1	X
830	BOLTS	95	142	49S	200	49S	3751	3751	8	N
			830	27K	12	27K	2007	2007	6	N
			830	23G	10	98Y	3751			
			830	98Y	36					

FIGURE 3.7. RELATIONS IN THIRD NORMAL FORM.

Normalised relations are a compromise between unnormalised relations and binary relations, in that, because of their basic nature, fewer relations in third normal form will be affected should changes occur in the application or data requirements.

The process of normalisation was originally defined by CODD (ref. 3) within the context of his relational data model. It is a concept which has been adopted in data base design irrespective of the data base philosophy used. It is a tool in producing a data independent model of data.

3.6. INTEGRITY.

Whereas different data base management software can support varying degrees of data independence the subject of data base integrity is a much more 'black and white' issue, a D.B.M.S. MUST ensure the total integrity of the data base. Integrity is the most important feature of a D.B.M.S. the data base must be timely, consistent and correct, otherwise the data base is no practical use. The data may be clerically irreplaceable and such a failure in an on-line environment can cause havoc in user departments.

3.6.1. Integrity Features.

There are 3 features of integrity:-

(1) Prevention of integrity failure.

(2) Detection and timely warning of integrity failure.

(3) Recovery including recovery of programs operating at the time of the integrity breach.

A data processing department must ask itself 4 questions when installing a data base:-

(1) How economically can a secure operating environment be established to support a data base?

(2) How to catch integrity breaches quickly to localise the problem and rectify with the minimum interference to other processing?

(3) In failure situations, how to select a recovery procedure which economically meets the desired recovery performance level?

(4) What price is to be paid in terms of operating constraints and additional processing overhead, in order to provide the basis for these recovery mechanisms?

Integrity means that data stored in the data base is accurate, timely, secure and private. This implies the data base is protected against theft, destruction, improper modification, unauthorised examination or deletion of data.

3.6.2. Integrity Controls.

Protection is need against data loss at all stages of processing:-

All data transferred between terminals or peripherals and processor should be checked by the operating system or preferably by hardware. If hardware failure occurs the user must know exactly what has happened and what data has been lost so that appropriate action can be taken.

There must be protection against erroneous update of the data base.

Protection is needed to prevent unwarranted interaction between programs.

With regard to these hardware and software aspects, the RELIABILITY of a system is defined as the system performing up to the required standard for a specified period of time. The AVAILABILITY of a system is the percentage of time the system performs as required. Both of these definitions can be applied to data base management systems as indeed to any other computer system.

To improve both factors, systems can have 'self-testing' procedures which can check for types of failure. Such mechanisms can be hardware or software and should be used after a specific time interval. An example of such a mechanism in common use is the error retry used after read/write operations. Error reporting can contribute to reliability by highlighting temporary failures so that the problems causing these failures can be corrected before permanent failures are precipitated.

The types of integrity controls fall into several categories.

Source Data Controls.

To ensure that data is correctly sent. This is done by 'registering' the input document at the point of entry or in the case of batch input, the maintenance of hash totals for the whole batch.

Source Data Validation.

To detect invalid codes, unreasonable amounts and other improper data values.

Processing Checks.

To ensure data base integrity, there must be as many processing checks as possible. There are several types of checks:-

Existence checks - does a value exist?

Accuracy checks - usually used in checking the validity of input data.

Combination checks - for example when a customer makes an order with a salesman, the details can be checked to ensure they relate to the branch office which is the source of the order.

Completeness checks - to ensure that input has the prescribed amount of data fields and that data is numeric or alphabetic as required.

Reasonableness checks - are used to determine if predetermined limits have been exceeded.

Self-checking codes such as check digits where the last character is derived from the previous string, is a method for identifying invalid input keying.

Programmed Checks.

Many programmed checks can be used, the objectives of which are:-

(1) The detection of the non-processing of data using record counts, control totals and hash totals.

(2) To determine whether arithmetic functions are performed correctly, i.e. using limit checks, cross footing and zero balancing.

(3) To determine that transactions are posted to the correct record.

(4) To ensure that all errors detected in processing are corrected.

Record Totals.

At the record level, the keeping of totals of the number of records added to and deleted from a file. These totals are checked against the transaction log to ensure the numbers of records are correct.

Development Standards.

The working practices of computer staff must also be secure in system design, programming and computer operating. One of the key aspects of maintaining accuracy in working practices is to ensure the timely update of documentation as changes occur to the data base or programs.

3.6.3. Implications of On-line Systems.

The updating of a data base on-line has many more integrity implications than batch updating. The basic problem is to provide sufficient controls over updating on-line without significantly affecting the updating performance. Some points to consider:-

(1) Is it practical to permit only one on-line program to update the data base at one time?

(2) Ensure that all programs under test only access a test data base.

(3) Store a duplicate copy of the data base off-site.

(4) Keep the data base as simple as possible.

(5) Write an audit trail of each transaction affecting the data base. The audit information should contain program number, terminal or operator name, time and type of update. A periodic analysis run of the audit trail can ensure integrity is being maintained.

(6) The establishment of change authorisation procedures so that all affected users have the opportunity to review proposed changes.

(7) It may be possible to split data base files over several cartridges to overcome problems with disc failure.

(8) Run control programs to check totals maintained at a terminal against those maintained at the mainframe.

(9) Security Certification which is:-

(a) A method of measuring, testing and evaluating the effectiveness of the system's integrity features.

(b) Determined by considering all potential threats to the security system, evaluating how well the security aspects of the D.B.M.S. anticipate, prevent, respond to and record potential security violations.

Such certification is covering the whole security aspect including both integrity and privacy areas.

3.6.4. Consistency.

Another aspect of integrity is consistency.

Consistency of part or the whole of a data base usually can only be defined by the user or owner of the data base. The data base as a whole at any instant of time is unlikely to be completely consistent because whenever any part of a data base is being changed, that part is likely to be inconsistent for an interval of time and should not be examined by other users until that part returns to its consistent state. If the time interval is too long performance of the system will be affected.

There are 3 characteristics concerned with maintaining consistency:-

(1) What quantity of data is involved?

(2) What is the size of the time interval?

(3) Is the user attempting to only read the inconsistent data or to update the data?

Consistent Update.

Whenever a program or subroutine code is being executed the particular program/subroutine is often referred to as a RUN-UNIT. Such run-units manipulate sub-schema records but the separate run-units must drive and control the consistency protection mechanism which should be provided by the D.B.M.S., this is because the D.B.M.S. can not define a consistent state. As a result, the run-unit must define the start and end of a consistent set of data base updates. The execution of the code between this start and end point is called a SUCCESS-UNIT. Consistent retrieval can be achieved simply by ensuring that part of the data base should remain available and unmodified for the duration of the success-unit.

Time Consistency.

The data in the data base is being constantly updated, however in producing reports the results would not represent the state of the data at any one point in time. The problem is that when several reports are produced at different times, based on slightly different data the reports may not balance. Time consistency is clearly very difficult to achieve. If such a consistency is important, each transaction should be given a time/date and the D.B.M.S. should attempt to produce consistent reports by:-

(1) Preventing updating while the report program is running, this is usually unacceptable.

(2) Copy data that is to be used for analysis, the copy time may prove an unacceptable overhead.

(3) To maintain a previous version of data marked with time/date currency i.e. chaining previous versions to present record. This means a heavy storage overhead, but nevertheless it does make historical report production possible.

The user should be able to specify to the D.B.M.S. consistency auditing procedures e.g. checking procedures for schema tables, indices and system pointers.

The reasons for consistency auditing are as follows:-

(1) Management has a legal obligation towards shareholders and government taxation agencies to ensure accurate financial accounts are kept.

(2) Data is used as the basis for management decisions, it is prudent to ensure that this data is kept properly.

(3) Errors and malpractices in administration.

(4) In automated systems, storage of erroneous data may initiate a chain reaction which can reach outside organisations causing losses and possibly bad publicity.

(5) If privacy legislation is introduced extensive auditing is needed.

3.6.5. Concurrency.

When two or more run-units are accessing the data base at the same time there is the possibility of requiring the same data. Multiple access to the same data must be permitted without erroneous interaction between users, alternatively, accesses to the same data are queued, but this will at times mean severe performance degradation. Concurrent access requires that the D.B.M.S. must be written in re-entrant code or each user must have separate versions which is wasteful of core and difficult to control. It is also helpful if the D.B.M.S. routines are relocatable, this gives the capability to vary buffer sizes and to vary the routines kept in main store depending on frequency of use.

3.6.6. Concurrent Access.

A number of choices are available when supporting concurrent access:-

(1) When one run-unit is updating the data base all other run-units may be locked out, or only those updating the data may be locked out.

(2) The lock-out may be until the update is complete or for the duration of the run-unit.

(3) The lock-out may apply to device, file, block or physical record or all records of a particular type.

Similarly locks may be applied in different ways:-

(1) By setting a bit against the record on the data base. This is inefficient because it requires a write to set the lock. There is also the problem that during the unlocking a failure may occur.

(2) The keeping of a list of locked records in main store in terms of data base keys. This involves searching before each access of the data base.

(3) The keeping of the whole locked record in main store. This has the same disadvantage as (2) but the D.B.M.S. may operate by keeping as many records in main store as possible, especially those frequently accessed, writing records when there is the least demand on channels or when the record buffers become crowded.

It should be possible to provide a time limit on locked units.

3.6.7. Interference, Roadblock and Deadlock.

Interference, roadblock and deadlock are three differing degrees of concurrency access conflict.

Interference (one writer and one reader).

When a writer updates the data base, the data record and pointer may need updating. If the reader then accesses the same record by the same pointer between the updating run-unit updating the record and the pointer then a malfunction will occur. Detection and recovery is straight forward but there will be a performance penalty.

Roadblock (one writer).

When writing to a set of records, all records must be put on to the 'lock list' before updating takes place. The 'lock list' is then a list of all records that can not be allocated for exclusive use. If another writer requests a record from this group, the request must be placed on a queue. The request is 'roadblocked' and it cannot continue until the required record is unlocked.

Deadlock (2 or more writers).

The most quoted example of the cause of 'deadlock' is when writer X locks record A and wants record B. Writer Y locks record B and wants record A. Writers X and Y are 'deadlocked'. One of the run-units must back track.

There are 2 types of control that can be exercised.

Exclusive Control.

If data is to be altered by a process, the entire sequence of events essential to alteration must be protected from any interference which would affect the proper completion of the change.

Shared Control.

This control is required for the process of reading data but not changing it. The process requires protection against any changes in the data which would affect the validity of the whole set of data the process requires.

3.6.8. Data as a Resource.

Data can be regarded as a resource to be allocated to concurrent processes.

Consumable Resource.

Once such a resource is assigned to a process it is consumed and is no longer available for re-allocation. Examples of such a resource are card images from card readers and messages or signals such as 'ready-to-send' 'ready-to-receive'.

Re-usable, Conserved Resources.

These resources can be re-assigned to another process when released by a using process.

A conserved resource is usually a physical resource which can only have one owner at any point in time i.e. the locking of a data resource is inherent in its allocation to a process. It is physically impossible for instance for a printer to output 2 lines of print simultaneously. Undesirable interaction of processes cannot destroy the integrity of conserved resource because simultaneous use is impossible. Alternatively if the internal state of a re-usable resource can be modified during use, it is a conserved re-usable resource only if it can be initialised on assignment to another task.

Re-usable Unconserved Resource.

The 'lock-out' of a process in the use of the data is not inherent in the allocation of data to a process. It is certainly inherent to the physical access mechanism resource but not to the data being accessed.

Unlike conserved resources, processes using unconserved resources must be distinguished by how they mean to use the resource. Processes that read data must still have option to 'lock-out' other processes wishing to modify it. A process which only wants to read data always has the option of reading, whether or not data is locked by another process. Data can be read concurrently by any number processes but can only be modified by one process at a time, thus it is impossible for a concurrent update process to make 'lock-out' decisions.

When data is allocated by the operating system to concurrent update processes it is usually at the file level, consequently if a process requires to update multiple files there is an integrity problem, otherwise if there is 'lock-out' of files imposed, there is a good chance of deadlock occurring.

For a D.B.M.S. such a level is unacceptable. The object of 'lock-out' is identified by giving named data items in named groups and records which satisfy a boolean solution expression.

In designing a D.B.M.S. the unit of data in terms of which object data is identified can be at the file level for coarse resolution or at the item instance level (value) for the finest degree of resolution.

If resolution is at the file level it is easier for deadlock to be prevented because a smaller number of resource units makes it easier for a process to declare aprioriits set of needed resources.

Resolution at the item level is the ultimate in sharing but results in more book keeping overhead. The possibility of deadlock is much reduced but it is practically impossible for a process to specify its set of needed resources apriori.

3.6.9. Lock-out of Concurrent Update Processes.

It is important to remember that update processes are not requesting full exclusive control of object data only exclusive control with respect to update actions.

In some D.B.M.S. implementations lock and unlock are implicit in the OPEN/CLOSE statements, but this puts an unnecessary and inefficient burden on the system. It is much more likely that explicit LOCK and UNLOCK instruction will be used within the application processing code. The LOCK instruction tells the system to test for the presence of a lock on the object data which may have been set by another concurrent update process. If the lock is set, the requesting process will be put into a WAIT state and the lock must be tested at intervals until the lock becomes unset. The lock request could be queued in First-In-First-Out (F.I.F.O.) sequence, it could be queued in a priority sequence or it may be possible for other processing to be done in the interim.

If the data base is divided into mutually exclusive units of data, a single bit lock can be associated with each unit of data. If however the identified object data is allowed to overlap with other identified object data then maintenance and testing of locks becomes more complex. Each identified object data unit would have to be kept in a current lock table and the table would require checking on the receipt of each new LOCK instruction. The process of checking, testing and setting of any locks on the object data must not be interrupted by another process checking, testing and setting locks.

Two different LOCK instructions are required in the case where a reading process locks out an update process while a second reading process comes along to read the same data, clearly there is no need to deny the second reading process access. This requires:-

(1) A READ Lock (LOCK-R) which is issued by a process reading data desiring that concurrent update processes be locked out but nevertheless permitting any number of processes which had issued similar LOCK-R locks to access the same data, and

(2) an UPDATE Lock (LOCK-U) which is issued by an updating process and which locks all processes out of the same data units.

Like so many other trade-offs in computing, the more sophisticated locking mechanisms may reduce the chance of deadlock but will equally reduce the system's performance and throughput due to the heavy locking mechanism processing overhead. Clearly a compromise solution is to achieve a balance of improved performance while permitting and accepting the possibility of deadlock. With such an acceptance of deadlock the D.B.M.S. must be able to either detect deadlock when it occurs or prevent deadlock occurring.

3.6.10. Detection of Deadlock.

The same method of detecting deadlock is used in D.B.M.S. when the resource is data, as in operating systems when the resource is the computing system resources.

The D.B.M.S. maintains a state graph showing resources allocated to each process at any point in time. The graph is updated each time a resource is allocated to or released from a process. Whenever a process requests a non pre-emptive resource (i.e. a pre-emptive resource is one which forces a process to release one or more of its other resources) a routine checks for cycles in the graph which would result if the request for the data resource was granted.

Figure 3.8. gives the graphic representation of the simplest form of deadlock. D.B.M.S. must control multi-process state graphs.

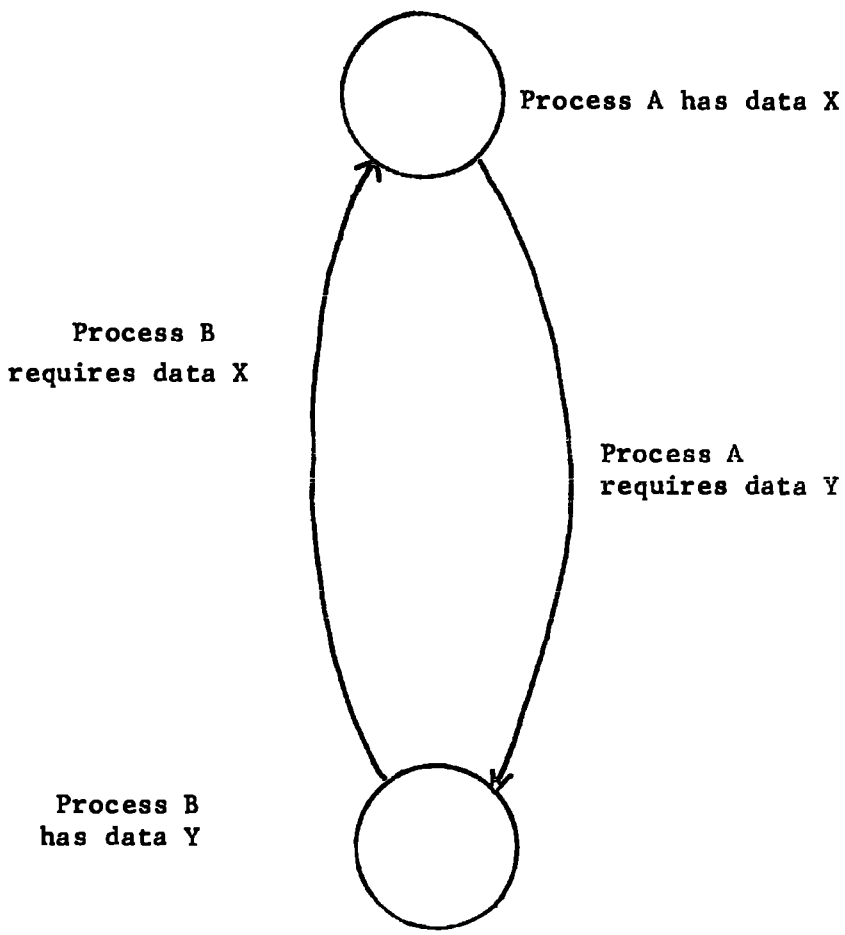


FIGURE 3.8. SIMPLE STATE GRAPH.

If a cycle would result and the requesting process has exclusive control of resources, one or more of the processes in the cycle must be backed out. The processes should back out in the least disruptive manner until sufficient

resources become available to remove deadlocks in the remaining set of processes.

To avoid the necessity of backing out a process in execution, deadlock must be prevented before it occurs. To be able to define the prevention criteria, the conditions of deadlock need to be known.

3.6.11. Conditions of Deadlock.

Lock-out.

A process can obtain exclusive control of needed resources. There is no solution other than to 'back out' resources of other processes that are not under exclusive control.

Concurrency.

2 or more update processes compete concurrently for exclusive control of 2 or more resources.

Additional Request.

A process can request exclusive control of additional resources while holding exclusive control of other resources.

No Pre-emption.

A process cannot be forced to release exclusively controlled resources before it is finished with them.

Circular Wait.

A circular chain of processes exists such that each process holds a resource which is being requested by the next process in the chain.

Deadlock can be prevented by relaxing any one of the above conditions independently. The 'lock-out' condition cannot be relaxed. However relaxing each of the other 4 conditions imposes particular disciplines on the use or operation of the system. Each condition is called PRESEQUENCING, PRECLAIMING, PRE-EMPTING and PRE-ORDERING respectively.

3.6.12. Prevention of Deadlock.

Presequencing Processes.

This is an acceptable method only in a batch environment where update runs can be pre-sequenced.

Pre-emptive Process.

The process is required to release all exclusively controlled data when subsequent requests are denied because of possible deadlock. Data resources cannot reasonably be considered pre-emptible when undergoing modification.

Pre-ordering Resources.

An arbitrary pre-ordering system is established on all resources which are non pre-emptive and used under exclusive control. All resource requests from each process must be issued according to pre-ordering i.e. a resource which is requested by a process must be lower in pre-ordering than resources currently held. This means that the circular chain of processes both holding and requesting resources is impossible. This method could impose severe and perhaps unnecessary discipline on programmers.

Pre-claiming Needed Resources.

Each process declares and receives exclusive control of all needed resources before using any of them. A process can never request resources while holding others i.e. a process must state apriori a maximum set of resources needed for the entire process. A less restrictive alternative is possible if the process can be partitioned into a series of steps so that all resources can be released or unlocked between steps.

An important point is that no resources are locked by a process until all resources required by the process are available for its exclusive control. Under such a strategy, a process may ask indefinitely for exclusive control of a particular combination of records. Such a system can be solved by the system adding 1 to a counter on each pass. When the count reaches some defined threshold, no other process can be given exclusive control of resources that are needed by the requesting process.

All the needed resources would be held in reserve until all became unlocked and available for exclusive control.

The effectiveness of this preclaim strategy requires a reasonably fine resolution be possible in identifying data to allocate exclusively to the process which implies that the object data unit is at least at the level of record.

The conclusion that can be drawn from this, is that if processes are to be run concurrently and if deadlock is to be avoided then all processes must state their resource needs apriori. To relax the preclaim means that competing processes cannot be run concurrently or that deadlock and the backing out after detection is possible.

3.6.13. Classes of Concurrency.

There are different characteristics associated with concurrency that are dependent on characteristics of run-unit, success-unit and the degree of data access.

(1) The case where there are any number of read-only run-units.
There should be no problem with concurrency in such a situation.

(2) One update unit of any type.
During recovery the data base is only available to this run-unit.

(3) One update, several reads.
Record level locks need only apply for duration of the D.M.L. commands.

(4) Simple update run-units with simple read-only run-units.
The data lock applies across the D.M.L. command but the sub-schema record must logically be locked from the time of retrieval to the point of modification (by the definition of a simple run-unit, only one sub-schema record is read).

(5) Any number of simple run-units of any type with a read-only run-unit having:-

(a) Long Success-units.
The D.B.M.S. cannot afford to freeze all updates for the duration. Another method is needed, possibly the retention of superceded records.

(b) Short Success-units (short or long run-units).
The lock out of concurrent updates is possible.

If the scope of the run-unit is static in extent, the consistency can be maintained by freezing part of the data base.

If the scope is dynamically changing:-

(a) it must be accepted that the run-unit results will be dependent upon the scheduling sequence of other run-units;

(b) it is possible that deadlock will occur;

(6) Any number of simple updating and read-only run-units with one or more general updating run-units having:-

(a) Long Success-units.
If the success-units are long, it becomes impossible to guarantee the required response time.

(b) Short Success-unit (short or long run-unit).
Lock out of data base becomes feasible.

(7) A long read only run-unit with concurrent updating run-units.

(a) Single updating run-unit with a long success-unit.
The read-only run-units are supported via retention of superceded record versions.

(b) Any number of updating run-units with short success-units.
This is the general case of high availability in on-line systems with possibly a concurrent batch reporting facility.

Conclusions from the above examples can be made. The scope and duration of success-units governs the probability of contention between concurrent run-units. The more precisely defined and the smaller they are the better. The most troublesome problems are caused by the processing of run-units with long success-units and other updating run-units.

3.7. PRIVACY.

The concept of data base security consists of protection against unauthorised access as well as protection against data loss. Indeed the main area of the management of data is concerned with data security, that is:-

(1) Protection against the inadvertent damage or loss of data causing a loss of DATA INTEGRITY (section 3.6.).

(2) Protection against the inadvertent or unauthorised disclosure of data, that is the loss of DATA PRIVACY (section 3.7.).

(3) The recovery of the data base and the restart facilities used in the case of failure. Should there be a failure to recover, this in itself would cause a break down in integrity (section 3.8.).

3.7.1. Aspects of Data Base Security.

The data base should be protected, reconstructable, auditable and tamperproof.

Users must be identifiable.

Actions with regard to the data base should be authorised and monitored.

Protection - is needed from fire, theft or other forms of destruction.

Reconstructable - data must be reconstructable because accidents do happen no matter how good the precautions.

Tamperproof - programmers and other data processing staff should not be able to by-pass controls.

Identification - users of the data base must be immediately identifiable.

Action Authorisation - such actions must be able to be checked.

Physical security is an important aspect of security. Door locks, guards, alarms and fire protection for data base files are as much a part of the data security scene.

Administrative controls are required to ensure the system is used correctly and to prevent mis-use of the system by data processing staff. A controlled, regulated and clean computer room environment is vital. The security must be checked out periodically by test procedures. Administrative controls extend beyond the immediate data processing environment to user departments, auditors and general management.

Loss of security can be attacked in 3 ways:-

(1) Minimise the probability of a security break happening at all.

(2) Minimise the damage if a security failure occurs.

(3) Design a method of recovering from damage.

3.7.2. Aspects of Privacy.

Within the field of data security, data as a resource becomes threatened by unauthorised and uncontrolled access to the data. Possible threats to such privacy are:-

(1) If data about a person is held against his authorisation and not controlled by him.

(2) Wrong data can find its way into the data base in 5 ways:-
   (a) Deliberate falsification.
   (b) Negligent or inadequate design.
   (c) Negligent or inadequate operation. In many respects this is most worrying because of the lack of intent on behalf of the offender.

(d) Unauthorised use of data in the data base. This is primarily a problem of control of access to the data base.

(e) Mis-leading use of data.

The forms of protection that are available:-

(1) Better control over data/system design in a data base environment.

(2) Improved control over access.

(3) Verification of data input.

(4) Control over the uses to which data is put.

3.7.3. Privacy Control.

Privacy is the most important aspect of concern to the layman in the whole area of data processing. Unauthorised access to the data base can be only prevented by imposing controls at several different points:-

(1) At the terminal by password or badge.

(2) Encryption at the terminal prior to transmission and the return to the normal coding when the data reaches the mainframe. Alternatively the data may be stored in coded form.

(3) At the storage device, check that the user can use the device, the wrong device may have been loaded by the operator.

(4) Restricted access at the physical level. It is possible to have access control lists for files, record types, data item types etc. If control is exercised at the occurrence level there would be a tremendous overhead. Such a facility may be provided by a user.

(5) At the logical level, authorisation is very important because it is at this level that the context of the logical access often determines whether access should be authorised. Separate physical files, one containing personnel information the other containing salary information

may not need protection, but if logical relationships were drawn between data in the two files then problems would arise.

(6) At the program level, a program may have a control list which limits execution to certain users. Similarly control may be exercised over system commands issued by an on-line user.

3.7.4. Types of Access.

The privacy restrictions that are imposed are dependent on how the data is used, such types are:-

(1) Extraction of data in several different ways:-

- outputting in detail;
- manipulation by operations without direct output for statistical purposes or summarisation, there is no output of individual records.

(2) Making changes to the data base - such changes can be identified by the data management language (D.M.L.) commands executed. Such modifications entail:-

- modifying existing data items;
- adding new data items to existing records;
- insertion of new records in a file;
- appending new records to the end of a file.

(3) The changing of access control lists - such lists can be associated with programs, devices or data types.

3.7.5. The Identification of a User.

Single user passwords do not provide adequate security in a data base environment. Passwords as character strings can be easily broken. When requests are made for certain key programs, data or storage media, privacy control may interrogate users as to their status e.g. by project number or additional password.

There must be pre-emptive facilities to access all data (not necessarily to retrieve actual values). This is to permit security checking to be made to ensure integrity is being maintained.

3.7.6. Data Ownership.

Control of access to data should be the responsibility directly or indirectly of the owner of the data. The user may not directly control the accessing procudures but he should be able to stipulate the level of control he requires to his data.

With the creation of new files it is often regarded that the user who created the file is the owner. In complex integrated data environments the ownership of data can be blurred.

In the not too distant future there will be the need for an access control language which will use boolean expressions to define the access control. Such a language may also specify what to do if the access conditions are not met. The likelihood is that such breaches would be logged, and:-

(1) the terminal would be locked to prevent further use;

(2) the user would be notified;

(3) the user would be advised as to how the access condition failed;

(4) send an error condition to the program;

(5) notify the administrator of the data base.

It must be remembered that the data base is only a part of the total picture, security of input and output is required. The print outs produced are as confidential as the data in the data base.

3.7.7. Access Control Mechanisms.

When considering access control mechanisms it was postulated by Schroeder that there are 4 criteria that can be applied to such mechanisms:-

(1) Functional.

The privacy mechanism must meet the protection of user needs in a natural non-restrictive manner.

(2) Cheap to Operate.

The mechanism must not cause a great machine overhead and should not cause an administrative nightmare.

(3) Simple.

The mechanism must be visible to give confidence in the viability of the system.

(4) Programming Generality.

It must be possible to vary the protection environment without causing programming problems like re-compilation.

Attaining all 4 criteria is impossible, the only possibility is to provide a trade off between flexibility, simplicity and economy.

The authorisation which is the result of the access control checking involves the comparison of restrictions placed on objects i.e. programs, terminals, data files or processes with the privileges of users that want access to those objects.

Access to data is only a subset of the authorisation problem which is closely associated with the particular hardware and operating system implementation.

3.7.8. Requirements of an Access Model. (Reference 4.)

Data owners must have the ability to specify the level of access.

The authority and privilege set that is operating on a process must be of a dynamic nature (i.e. as events are occurring privileges should not remain static). For example a user U may wish to start a highly classified job from terminal T. The job may access classified files, create output, call other processes all of which may have different levels

than user U or terminal T. Normally the user would need the capability to access the highest level of object in the computation, this implies that the authority of U and T must be greater or equal to the authority of any other object in the computation including the created output. With an adequate protection system, there is no reason why this should be. A system is required to allow the specification of authority of all objects that fall within the domain of a single process. An object can be a file, record, data element, memory page, interrupt call, terminal, user access key or process itself. The mechanism should be able to deal with the fact that terminals can obtain different clearance attributes at different times.

The definition of security requirements must be calculated on the basis of the characteristics of data or the context of a relationship. The contents of data could be vital to the authorisation of access, for example a payroll clerk may only be able to access salaries of less than £5,000. A user may have authority to access specific data while performing one job but not while performing another. An adequate system will permit the specific definition of such privacy relationships in an easy and efficient manner.

The concept of distributed data has many implications on the requirements of an access model.

Authority must be specified down to the element level. Record or file level is not sufficient. This will cause greater overhead but it should be possible to reduce the problem by more sophisticated hardware/software implementations. Whereas in the past authorisation was controlled by bits or tables of data but now the tendency is for such decisions to be made by programs.

3.7.9. The Privacy Environment.

Data processing environments are becoming complex with resource sharing, remote and local users and batch and time sharing operations.

The Security Structure.

Security Objects - these are the basic entities in the system that must be protected i.e. data items, records, terminals, programs.

Protection Groups - a set of objects that have the same protection attributes. The group of which an object is a member determines how the object is protected.

Groups can be defined in 3 ways:-

- levels.
- categories.
- or combination of the two.

Levels are hierarchical groups.

Categories are best described by an example. A customer file may have some data elements accessed by the Billing Department, these data elements are in a category.

Combining the two types gives something like

'TOP SECRET, NAVY ONLY'

the object is classified at the 'TOP SECRET' level to be accessed by the 'NAVY ONLY' category.

Therefore a protection group at level '1' and category 'k' are described in the couple (1,k). This takes only 1 value by the concatenation of 'k' with '1'. This is called the AUTHORITY or CLASSIFICATION OF OBJECT.

A data object is assigned to a protection group on the basis of the information content of the data, e.g. all salary records acquire membership on the basis of the record's general content but salary records with entries greater than £5,000 can be separately protected due to the value of this attribute.

Another basis on which data objects are grouped is by contextual sensitivity i.e. individual records may not of themselves be sensitive but if certain other data is available, the complexion changes. The grouping of such objects could prevent the use of component objects.

3.7.10. Security Attributes.

Certain security attributes are associated with each data object in the system. The attributes of object (i) are:-

The Classification of the Object (C_i).

The protection group to which the object belongs. This is expressed in terms of the component level (l_i) and category (k_i).

i.e. $C_i = l_i^c, k_i^c$

The Authority of the Object (A_i).

The authority specifies the protection groups to which the object has access. It is made up of the same components as the classification.

i.e. $A_i = l_i^a , K_i^a$

The Access List (L_i) associated with each object 'i' - the list is used to authorise other objects access to 'i'. There will be a variable number of entries, each containing:-

- an access clique.
- a privelege list.

The Access Clique - the name of a specific object or of a clique of objects previously defined in the system. Objects will be users or programs, with each object or clique having a unique system name which is system generated.

The Privilege List - a bit string of 2 bits for each privilege that might be granted. Privileges include such actions as read, execute, update, create or delete. The first bit indicates the privilege is granted to the clique, the second bit grants the clique the right to pass the privilege to some other object provided other requirements are met.

Each ordered pair (access clique, privilege list) is called a CAPABILITY. This capability is the medium to pass access privileges from one object to another.

3.7.11. Implementation of the Access Model.

The objects of access are programs (p), data (d), terminals (t), jobs (j) and users (u).

For one object (i) to gain access to object (j), the following conditions must be satisfied:-

(1) The authority level of (i) must be greater than or equal to the level of (j):-

$$\text{i.e. } l_i^a \geq l_j^c$$

(2) Categories authorised to (i) must be at least all of those in classification of (j):-

$$\text{i.e. } k_j^c = k_i^a \cap k_j^c$$

(3) The authority of (i) is greater or equal to the classification of (j)

$$\text{i.e. } A_i \geq C_j$$

(4) The accessing object must be on the access list of the accessed object.

$$\text{i.e. } (i) \in L_j$$

this permits (i) to access (j) with privileges listed in the 'i^{th} L_j' entry.

If these criteria are applied to the activities of the system, of particular interest are:-

- gaining access to the system;
- invoking an existing program (process);
- the calling of 1 process by another;
- program accessing a data object;
- creating a new object;
- deleting an object;
- changing the access list of the object.

As an example, when a user attempts access at terminal (t), the following must be checked to hold:-

$$A_u \geq C_t$$

$$U \in L_t$$

If access is granted job (j) is created. The attributes of (j) will dynamically change during the process, at creation the job takes the authority of the user:-

$$\text{i.e. } A_j = A_u$$

$$C_j = \emptyset \text{ (NULL)}$$

The job has no classification at first, the situation changes as objects are accessed.

After the job creation a stored program (p) is usually invoked. The requirements are the same:-

$$A_j \geq C_p$$

$$U \in L_p$$

The program may have inherent classification due to sensitive code. The user must also have been granted 'execute' privilege in 'L_p'.

When the program is invoked, the job will change classification. If the program carries a higher classification than the job then:-

$$C_j \longleftarrow \text{Max } (C_j, C_p)$$

This causes C_j to always reflect the highest classification associated with that job, giving rise to what is called the 'high water mark' attribute that can be passed to created data.

The called program may have a certain authority 'A_p'. In general a program will have no authority but will have some authority granted by the calling object, in this case the job.

Program 'p' may call other programs i.e. 'P_2, P_3....'.

The requirements are identical to the case where the job calls a program except the program name itself may be an entry to the called program's access list:-

$$A_{p1} \geq C_{pn}$$

$$u \in L_{pn} \cup P_1 \in L_{pn}$$

i.e. some programs (or objects) may only be accessible through other programs. User (u) might not be on the access list of 'P_n'.

When 'P_n' is called, the attributes of 'P_1' (and therefore (j)) must change:-

$$C_{p1} \longleftarrow \max (C_{pn}, C_{p1})$$

$$C_j \longleftarrow \max (C_{p1}, C_j)$$

'P_1' can grant certain authority to 'P_n'.

Programs will need to access certain data objects, the requirements are:-

$$A_p \geq C_d$$

$$u \in L_d \cup p \in L_d$$

After granting access:-

$$C_p \longleftarrow \max (C_p, C_d)$$

$$C_j \longleftarrow \max (C_j, C_p)$$

If the file is written (assuming that the proper privileges are granted):-

$$C_d \longleftarrow \max (C_j, C_p)$$

This 'high water mark' can occur when new data is created in the system.

Turning to the deletion of objects, if object (j) is deleted by object (i), (i) requires that access to (j) be granted with proper privilege.

Changing access list of object (j) by object (i) requires a very high CONTROL privilege.

This is a general concept of an access control system which should serve a multitude of users. However such a system cannot control contextual sensitivity.

To test for this sensitivity the system must be able to test a list of characters logically. For example:-

```
    If user name = NIXON
  terminal number = 64
              day = FRIDAY
  then access
          rights = PRINT
```

The system must be able to express 'LOGICAL AND', 'LOGICAL OR' and 'LOGICAL NOT'. In this way access to data can be controlled on the basis of data content:-

salary $\geq$ £10,000 access = NULL

Restrictions should also be based on the context of use i.e.:- 'salaries are not to be printed with names.'

If the above mechanisms are insufficient the user should be able to add on his own access control procedures.

3.7.12 Levels of Access.

There are 4 levels of access to data from a file to a user:-

(1) N - Null - no access is permitted.

(2) M - Manipulate - user may or may not be permitted to see the results of the manipulation.

(3) S - Statistical - use of data to obtain means, deviations etc.

(4) P - Print - the system will disseminate information.

Each lower capability is considered a subset of the next higher capability.

There is also a hierarchy of capabilities from a user to a file:-

(1) N - Null.

(2) A - Append - the ability to add information to the logical end of file.

(3) W - Write - the ability to change the existing information in a file.

(4) C - Change Access - the ability to change access restrictions on a file.

There are 2 other capabilities associated with each file:-

(1) O - Ownership - this capability implies that the user caused the file's creation.

(2) AC - Acquired - the file contains information not originated by the owner.

Ownership implies the user creates the file and can delete it.

Originator of information is the person who enters the information into the system as raw data having all rights of ownership and access changing.

The originator of information has control over the dissemination, the person can release copies of his data. Any file derived from an acquired file is also acquired.

3.7.13. Identification of Users.

There are several different methods of identifying users:-

(1) by user name;

(2) by password for logging in; some files may require extra passwords before accessing specific sensitive files;

(3) by program name, defining the program being used to access the data;

(4) by project name;

(5) by time of day;

(6) by day of week.

3.7.14. Computation of Access Rights.

The originator of data can provide a list of constraints over the access of his data. Such a list of constraints is called an 'Access Control List'. This list determines the set of requesting entities which have the given level of access.

In the type of system where constraintis based on content or context of data elements then it is insufficient to permit protected data to be manipulated and then to permit examination of the final result. If for instance a list of all people with salaries over £5,000 is extracted, the final list of names must be protected, but with access control, the sensitivity cannot be determined by simply examining the final result, access control must examine the change in access privileges at each interim step in a process. Because of the content and context constraints, access control must be at the data item level.

3.8. RECOVERY.

A reliable, comprehensive recovery system is fundamental to a successful data base system because the data will become the foundation of the business organisation.

Recovery is required when:-

(1) Part of the data base is destroyed.

(2) Machine errors occur during the transaction.

(3) An incorrect program contaminates the data.

(4) A user enters incorrect data, detects immediately and wishes to make a correction.

(5) An error is discovered later and all following dependent transactions must be corrected.

There are usually 2 different types of recovery required for batch and on-line updating of the data base. In an on-line system, the recovery goal is to bring the system back to the point of failure, i.e. no transaction is processed twice and the data base is reconstructed as it was before the failure. When an application program fails, any file updates performed by the on-line transaction that caused the failure, must be backed out. In a batch environment when the hardware or supporting software fails, updates from the file update log are applied to the latest copy of the files.

3.8.1. Logging.

Different types of data can be logged by a D.B.M.S. Input messages, output messages, file updates and 'end of transaction' records are all different types of data used in different circumstances to recover the data base.

The 'end of transaction' record when logged can indicate the end of each on-line transaction and whether the transaction was successful or not. When a file is updated, 2 different types of record can be stored in the file update log:-

(1) An image of the data base record can be stored in the log BEFORE the update takes place (BEFORE IMAGE).

(2) An image of the data base record can be stored in the log AFTER the update takes place (AFTER IMAGE).

A file update log should include the following data elements:-

(1) The transaction number.

(2) The source terminal identifier.

(3) The date and time of change.

(4) The application program identifier.

(5) The type of change.

(6) The before and/or after image of the changed record.

(7) The end of transaction record.

The message log containing details of the input transaction and possibly the output message, should include:-

(1) The transaction number.

(2) The source and/or destination terminal number.

(3) The date and time of the message.

(4) The message itself.

All logged data should be stored on 1 disc and 2 tapes. The disc log can be used for on-line recovery and the tape logs for batch recovery. The second tape may be used as back-up to the first tape.

3.8.2. Recovery Procedures.

This section describes some of the recovery procedures in common use in the early 1970's.

Back Up by Complete Journal.

Step 1 - Periodically all data base files are copied to magnetic tape or disc. The periodicity is a function of the amount of data to be copied and the time taken to copy. It is also related to the importance of the data and what other forms of back-up are available. Such a dump would be taken after several working days but probably at an interval of less than a week.

Step 2 - Every update made to any file during that period is recorded in another file, called a journal, which includes records added, modified and deleted. An 'after image' is taken of the data file, the index and any inverted lists.

Restoration - The whole file dumped in the first step is restored from the periodic copy. The journalised updates to the file are restored in chronological order.

This procedure enables the file integrity to be restored up to the last update prior to the system malfunction. It is operationally expensive because it adds an additional file write to every update operation.

Selective Dump of Updated Files.

Step 1 - As in method 1.

Step 2 - Whenever a file is updated, a file update flag is set in the D.B.M.S. directory.

Step 3 - Periodically the files that have been updated are dumped to magnetic tape, accumulating a succession of dump tapes.

Restoration - If during a system breakdown a file is lost, the file is reloaded from one of the dump tapes taken in step 3. If it does not exist, the file can only be taken from the dump obtained in step 1. The quality of back-up is not as good as that in method 1 and the overhead can be greater because a whole file is being dumped rather than individual records. The advantages are that the file dumps can be done at a low point in system load whereas in method 1 the dump must occur at the time of update.

Selective Dump of Updated Records.

This method is equivalent to method 2 but it is performed at the record rather than file level.

Step 1 - As in previous methods.

Step 2 - When a record is updated, a record flag is stored in the directory. The update may affect other records and flags must be set for these.

Step 3 - Periodically all records with flags set are dumped to tape. These can be merged with the full dump records.

Restoration - As method 2.

3.8.3. Recovery Procedures in Transaction Oriented Systems.

The major difficulty is the multiple update problem when a system crash occurs there is a 'ragged edge' of several transactions in different stages of completion.

File Restoration to last Complete Transaction.

Back-up is by the 'Complete Journal Technique'. When a transaction is completed, the journal is marked. Recovery is up to the last complete transaction only. An indication must be given to each of the terminal operators that the current transaction must be re-entered.

Back-up Journal with No File Restoration.

In this method only 'before images' log records and 'end of transaction' records are journalised. All records updated since the beginning of the transaction are put back to their 'before image' context. The current transaction is then completely reprocessed.

If both system and file recovery are required both 'before images' and 'after images' are required. 'Before images' are required for system recovery while 'after images' are required for file recovery.

Record Stamping.

In this method there is no back-up journal with the consequent reduction in file transfer overhead.

Step 1 - Each new transaction is given a unique stamp, normally a concatenated character string made up of the date and transaction number.

Step 2 - The input transaction details are written to the message log.

Step 3 - Each record updated by the transaction is stamped.

Step 4 - The 'end of transaction' record is written to the message log and the time stamp is updated in the terminal control record.

Restoration With Input Transaction Journal.

Step 1 - The control record containing the last update value of the stamp is read.

Step 2 - The last record on the input transaction file should have the same stamp value on the control record. If this is not the case the failure has occurred between steps 1 and 2 of the back-up process (between the increment of the stamp and the writing of the input transaction record). If the values are the same, the crash occurred during steps 2 and 4 of the update process.

Step 3 - Only if the 2 stamp values are the same can there have been any update processing performed. In which case the transaction is 'mock reprocessed', that is to say, the processing of the transaction is repeated without causing any of the records that have been updated in the original processing to be re-updated. When a record is found that does not contain the current stamp value then processing continues from that point and the records are actually updated. There is a complicating factor in the 'mock reprocess' in that running totals must be maintained during this repeated processing because much of the direction of processing may be dependent upon the original values on file. These values must be obtained by using the input transaction data to process the output values to obtain the original input values.

Step 4 - The user is notified that the last transaction has been completed so that he/she can continue normally.

If restoration must be performed without an input transaction journal then at step 2 above the terminal user must be requested to re-input the data so that reprocessing can take place. There is an added difficulty however should the user fail to re-input the data correctly, this could lead to incorrectly derived original data base values during the 'mock reprocess'.

3.8.4. Types of Failure.

There are many different reasons for the need to recover.

As many as 17 different types of failure identified by the Customs Early Implementation System in the U.S.A. (reference 5):-

(1) Application program.

(2) File fail but with file manager O.K.

(3) Operating system failure.

(4) File manager failure.

(5) Transaction processing exec. failure.

(6) Contaminated files.

(7) Disc pack failure.

(8) Disc drive failure.

(9) Disc/tape controller or channel failure.

(10) Log device failure.

(11) File back-up device failure.

(12) Communications hardware failure.

(13) Modem failure.

(14) Terminal control failure.

(15) Terminal failure.

(16) Communications line failure.

(17) Line printer failure.

3.8.5. System and Program Failures.

There are several different results of system and program failure.

Run-Unit Exceptions.

Events causing the premature termination of a run-unit:-

(1) Deadlock with other run-units over resource usage.

(2) Program error with fault detection at run time.

(3) Program or user requested termination.

Such faults are contained within the run-unit. If faults have occurred on the data base they should not be accessible to run-units.

Software Failure (of operating system or D.B.M.S.)

The data base and recovery files should be secure, however core tables of run-units, record locks and pending updates may be lost. The in-core recovery status may also be lost.

C.P.U./Hardware Failure.

The data base should be secure.

Data Base Store Media Failure.

Everything in the software failure category is safe with the exception that all or part of the data base stored on the failed unit(s) will be lost.

Corruption caused by a Previous Run-Unit.

This type of failure is one of the worst types when a run-unit appears successful but later access to the data base identifies a corruption. The problem can be further sub-divided into:-

- corruptions detected by subsequent run-units;
- user detected errors.

Causes of such a failure can be D.B.M.S. bugs or operational/user input errors.

3.8.6. Solutions to Program Failures.

Several solutions are possible in the event of a program failure:-

(1) The data base can be reloaded to the point where the program began and re-run if the fault is considered transitory.

(2) Restoration can be made to the point of the most recent checkpoint and the program can be restarted once again assuming the fault to be transitory. A checkpoint is taken to mean a dump of part or all of a data base file. The interval between checkpoints is determined on

the basis of time or updating activity.

(3) Program updates to the data base can be backed out using 'before images'. The data base has then been returned to the original state either at the point at which the program began or to a program restart point.

The problem with 'roll-back' as described above is that other programs that are being run at the same time will be affected. This is the case for batch as well as transaction programs and not only are update programs affected but also retrieve-only programs may have read incorrectly updated records which cause wrong decisions to be made.

Another important decision to be made is how far to roll-back a program? It may be difficult to permit automatic roll-back in a fairly integrated data base environment, with many transactions in the system, with each transaction having a logical sub-schema which has access to data on several different data base files each checkpointed at different time intervals.

It is particularly useful in a transaction-only environment, when there is a low hit rate on data base files, to take a system-wide checkpoint. When such a checkpoint is called, transactions will be held up until no activity is in progress. Depending on the duration of each transaction and the frequency the checkpoints are taken then these system-wide checkpoints have a minimum impact on response time and throughput. If one or more batch programs are involved the system-wide checkpoints would be impossible. Certainly if response time requirements are modest and single threading only is used in the T.P. monitor and D.B.M.S. then there is protected update which relieves the checkpointing synchronisation because each transaction has protected use of the data base and could be rolled back independently.

3.8.7. Solutions to System Failure.

The important difference in system failure from program abort is the loss of the operating system and main memory contents. Apart from data stored in the log file, in the data base or known by the operator, there is no other information available at the system failure. Otherwise conditions are as for a program abort. Currently successful data base recovery from system failure is dependent on human diagnosis and action. One problem is that if the log records are assembled in memory for blocking. The only solution to this problem is to 'force write' the log record so that it is immediately written to the log file.

Another problem is that if when a failure occurs the transaction processing has closed the data base but the T.P. monitor does not think that the transaction has finished, this may cause the data base to be updated twice in recovery because with the data base file closed the update may not be backed out. Such problems require human intervention.

It may be quite possible that while part of the data base is being recovered by the recovery utilities, the rest is accessible for normal processing. Clearly the integrity/recovery question relates to an installation's own environment and application requirements. The need to establish appropriate standards and procedures is clear.

Certain points that are worth remembering are:-

(1) As many realistic failure conditions as possible should be thought of in advance and the necessary recovery mechanisms tested (e.g. failure during recovery).

(2) If it is at all possible do not update on-line and do not update in on-line and batch modes concurrently.

(3) Think through the processing intent mode requirements for each program and avoid shared update mode unless performance demands it.

(4) Guard against 2 data base managers updating the same data base concurrently.

(5) Transactions should be kept short and the data base can only be closed at the end of a transaction.

(6) The effect of integrity mechanisms on performance should be closely examined.

(7) Find the optimum back-up copy frequency. Large data bases may require a segmented copy strategy.

(8) Enforce checkpointing standards, in particular for batch programs running concurrently with on-line programs.

3.8.8. Fallback Facilities.

It is most desirable that the D.B.M.S. provides a 'fail-soft' facility i.e. when a component of the data base environment fails some form of degraded service is possible. 'Fail-safe' is the minimum standard provided by a D.B.M.S. i.e. no data is lost when the system fails.

It would also be useful if the terminal access software or D.B.M.S. provided facilities to collect data input at a terminal which could later be processed by fallback batch facilities when normal service is resumed. This is usually only of benefit in an on-line data capture system. Such fallback procedures should be well documented and terminal operators should be thoroughly trained in their use.

3.8.9. Levels of Recovery.

There are 4 levels of recovery response, different applications require differing levels. Within each run-unit even different levels are required dependent on the exception. 'Cheap' recovery should not pay 'high insurance premiums'.

(1) Immediate Recovery - recovery time is unnoticeable. Such a recovery level requires hardward redundancy.

(2) Immediate for Data Base Media Failure only - the requirement here is for the concurrent maintenance of dual data base copies together with recovery file logging.

(3) Rapid - recovery time for run-unit exception not a great deal longer than normal response time i.e. within a number of seconds.

(4) Slow - equivalent to the time taken to reconstitute a significant portion of the data base. Probably several minutes.

3.8.10. Levels of Processing Recovery.

The run-unit recovery can never be more complete than the data base recovery and it is often chosen to be less.

The levels are:-

(1) The program is unaware of recovery taking place.
- instruction retry;
- input/output retry;
- D.M.L. command roll-back and retry.

(2) The program automatically recovers with a resumption at the start of the current success-unit. The following procedures are performed:-
- check failure type for recoverability;
- reset working storage;
- resume processing.

(3) Operator controlled restart with the following steps:-
- data base recovery;
- select corresponding program status checkpoint;
- restart program in restart mode.

(4) Manually controlled resumption from the beginning.

The restart of a program consisting of a series of success-units needs considerable synchronisation of:-
- data base state;
- program state;
- working storage contents;
- non data base files.

Terminal users must be aware of the state that they are in at such a restart point.

There are 2 types of recovery point:-

(1) A run-unit recovery point on either run-unit or success-unit boundaries.

(2) A system recovery point where no run-units are active and the system fail causes recovery at a convenient set of run-unit recovery points or to a complete system recovery point which precedes any run-unit recovery action.

3.8.11. Units of Recovery.

Normally physical stored records are regarded as the units of recovery, that is, the unit of data that is copied to an update log record. However there are several potential recovery units:-

- data item;
- logical record;
- total sub-schema data;
- stored record;
- page (or unit of peripheral transfer);
- an indeterminate sized storage area.

The smaller the unit the more intricate the recovery but a minimisation of the effect on normal D.B.M.S. performance. If the recovery unit is too large there could be severe concurrency problems.

3.8.12. Alternative Recovery Strategies.

Roll-Back.

It is the removal of debris by over-writing recently modified units of the data base with copies taken before updating (before images).

The basic requirements for roll-back are:-

(1) An accurate knowledge of the state of the data base and run-unit(s).

(2) An accurate 'before image' file.

(3) Identifiable run-unit recovery points.

The likelihood is that a hierarchy of roll-backs will be employed involving the D.B.M.S. only, then the run-unit only.

Useful roll-back types:-

Command Roll-Back.

If the D.B.M.S. meets difficulty executing a D.M.L. command then it must reset the data base, its own, and the run-unit's state to the start of that command without affecting the course of excecution of the run-unit.

This type of roll-back is particularly useful to the D.B.M.S. when:-

(1) recovering from transient errors;
(2) resolving contentions.

Success-Unit Roll-Back.

Restores that part of the data base being accessed within the success-unit of one run-unit to a consistent state. Because of concurrency controls, the intermediate results of the success-unit are not available to other run-units.

Run-Unit Roll-Back.

Several success-units may require to be rolled back. Data elements altered by this run-unit may already have been used by other run-units.

System Roll-Back.

Involves the roll-back of all success-units for all run-units.

Automatic success-unit restart after roll-back depends on:-

(1) The ability to recover working storage and non data base files to a corresponding state.

(2) The likelihood that the failure will not recur.

For short success-units, roll-back gives the quickest type of recovery.

Roll-Forward.

Roll-forward consists of:-

(1) Recovery of 'before images' of the entire or part of the data base.

(2) The synchronisation of the data base state with the system recovery point in the recovery log file.

(3) The application of 'after images' for all run-units to that part of the data base up to the end of the recovery file.

(4) System roll-back execution providing the latest possible consistent state.

(5) Alternatively, 'after images' may be applied for specified run-units up to a known safe point.

Restore and Reprocess.

It may be preferable to reprocess transactions from a transaction log especially where faults in the data base were caused by complete run-units. It may be good to have a log available for editing before reprocessing.

Restore and reprocess and roll-forward are generally incompatible and they cannot be used in conjunction because they react differently to altered transaction sequences.

Log Analysis.

There is no way of recovering from subtle corruptions which have been used by other run-units. End users must participate in correcting the data base in such circumstances, however an analysis of the appropriate log files may be helpful.

Analysis by record occurrence showing runs participated in and alterations made will be valuable.

Instead of making a run-unit restartable it may be better for each run-unit to have an abort routine. Failure can then be analysed to indicate what type of failure as follows:-

(1) transient;

(2) storage corruption;

(3) deadlock;

(4) requested termination;

(5) permanent fault.

It may be possible to try to restart where the failure type offers a reasonable chance of success.

3.8.13. Safety Precautions.

In recovery, prevention of need for recovery is better than the recovery cure, and the recovery may be swifter if certain conditions are forseen.

An organisation must decide how to augment the D.B.M.S. facilities to meet their own individual needs. This process consists of establishing limits for behaviour, determining requirements of application systems and ensuring the basic operating environment offers sufficient capability resilience to meet their needs.

Precautionary decisions that should possibly be made:-

(1) What classes of run-unit should be allowed?

(2) What level of concurrency to permit?

(3) Which recovery facilities of the D.B.M.S. to use?

(4) The policing of run-unit behaviour:-

- time out;
- D.M.L. command types and sequences permitted;
- success-unit scope limits.

(5) Which recovery techniques to adopt?

(6) Which recovery file types:-

- before images;
- after images;
- transaction journal;
- logging unit size;
- time span covered.

(7) The level of data base redundancy.

(8) What logging security?

- dual log file;
- remote log files.

(9) What level of hardware redundancy?

Some of the characteristics of a data base environment that add to the problems of error detection and recovery:-

(1) minimal data redundancy;
(2) undefined data ownership;
(3) complex pointer structures;
(4) an excessive amount of on-line updating.

If bugs occur in the data base then they should be fixed with patches, while there should be releases for new development.

The problem with unreliability of a D.B.M.S. is the same as any other complex software system i.e. the probability of the whole remaining fault-free is the product of the probabilities of all its component parts remaining fault-free.

The D.B.M.S. should provide extensive facilities to enable examination of any main memory areas within the D.B.M.S. Such a facility may be particularly useful should the recovery system fail.

3.9. PERFORMANCE.

A D.B.M.S. is a system overhead in the same way, if not to the same extent, as an operating system.

Performance is reduced in comparison with conventional systems by facilities such as data independence, complex data structures, data locking, privacy and recovery mechanisms. In the past the overhead was unacceptable, preventing the widespread use of D.B.M.S. The reducing cost of hardware and associated power has meant that the overhead can be made more justifiable and thereby more acceptable.

3.9.1. Performance Evaluation.

In comparing D.B.M.S. against conventional systems, it is difficult to make comparisons because D.B.M.S. advantages are largely intangible and comparing performance with cost is not strictly relevant. Some of the intangible benefits are:-

(1) increased programmer productivity;
(2) reduced amounts of redundant data;
(3) ease of meeting user requirements;
(4) data more accessible to the users as an information source.

With the introduction of a D.B.M.S. there is the likelihood of an increase in hardware requirements:-

(1) Main store requirements will increase but application program requirements will decrease due to the removal of file routines.

(2) Data storage is increased by pointer and index storage but reduced because of data redundancy.

(3) Additional channel capacity is needed because of application independence so that the percentage of relevant data in each physical block transferred is much reduced.

(4) It is likely that an increase in processing power will be needed.

3.9.2. Performance Optimisation.

Performance can be varied with the following circumstances:-

(1) With more main store, the system can have more run-units in store at once, more data can be held in buffers reducing access to the data base and the system may keep more of its routines in store.

(2) The choice of storage structures affects performance. It is a function of the volume of data, its volatility and the frequency of enquiries.

(3) Physical storage in the data base is the concern of the D.B.M.S. When accesses to overflow areas reach defined levels, the reorganisation of that part of the data base should occur. The D.B.M.S. should also provide data compaction facilities.

3.9.3. Tuning Data Base Systems.

Data base systems must be tuned before they are implemented. Inefficiencies arise because of poor scheduling, unbalances channels, device allocation, choice of inappropriate operating system parameters, programmer mistakes etc.

The most significant performance attribute associated with the D.B.M.S. is the data base design itself. The more complex the structuring and access, the more scope there is for tuning, and the harder it is to select the optimum tuning solution.

3.9.4. Statistics.

An organisation requires statistical feedback from a D.B.M.S. There are 2 types:-

(1) Static 'snapshot' data at any given instant in time.

(2) Dynamic data collected over a period of time.

The static type may be produced as a report using 'running counts' which are maintained during the normal utilisation of the D.B.M.S. The dynamic statistics are obtained by monitoring a particular resource or activity. This monitoring may be under operator control or at a pre-determined time interval.

Usage statistics that may be required:-

(1) Device, Channel Activity - this indicates whether capacities are adequate or if throughput is limited by a lack of direct access devices. Such statistics are usually provided by the operating system already.

(2) Amount of store used by core tables, buffers, modules and application programs.

(3) The frequency of access to/update of data elements. It is possible to measure only data element types not occurrences because of the storage overhead. The program itself would have to take occurrence statistics.

(4) The frequency of call to modules and the length of use. This gives an indication of which routines should be core resident.

(5) User error rates may be measured to analyse input error rates so that education can be given to reduce the rate.

There is the need to have available simulation routines to test the effects of various possible changes before actually implementing such changes.

There should be a complete audit trail of all activity on the data provided automatically by the D.B.M.S. Such a report should provide a complete history of processing and in the area of testing or fault finding the report would be an invaluable diagnostic listing.

3.9.5. Restructuring and Reorganisation.

Data is changed by application programs updating the data base. The content and definition changes directly or indirectly by the process of evolution.

These changes come in 3 groups:-

Restructuring.

This causes changes to the schema and as a result probably changes to the physical data base. The possible changes could be to:-

(1) add or delete fields in existing records;
(2) group existing records into different data structures;
(3) change indices and access paths;
(4) re-arrange the storage of records of a given type based on a new addressing algorithm.

Generally such restructuring requires the data base to be dumped and then restored in the new format. The implementation of restructuring require considerable study to make the process more flexible and less time consuming.

Garbage Collection.

This is the consolidation of free space left by deleted records into 1 contiguous area of storage.

Reorganisation.

The arrangement of records between blocks to improve efficiency by:-

(1) Moving records from overflow areas to free space in prime data areas.

(2) The distribution of records evenly across an enlarged prime data area, to leave adequate free space to store new records.

(3) The re-arranging of records to ensure their physical proximity to other records that relate to them logically.

Reorganisation can reduce peripheral transfer time by allocating data to a storage device to take account of the nature of queries.

There are 2 important points that have arisen from studies made into disc accessing (reference 6), they are:-

(1) Half the total activity of the data base is confined to 10% of the data base. The time the reading head spends in motion is reduced to a 1/3rd. of the value if the file is uniformly accessed. As the data base ages the 10% of the data base that constitutes half the total activity would increase. This needs monitoring so that it can be reorganised as necessary.

(2) Such placement does not affect the other 2 components of access time:-

- Latency - the time taken for the required data to come under the read heads.

- Initial jump time, which is the time associated with the fact that the read heads have to move at all.

These 2 times together account for 75% of the total access time.

The initial jump time accounts for approximately 60% of the total seek time. By attempting to cluster records on the storage device the probability of having to seek to execute the next access request is reduced.

Further gains may be made in performance by scheduling access requests. In any single program request for data there may be several physical data access requests generated, several of which are likely to cause read heads to move backwards and forwards across the disc's surface. The average jump if there is no scheduling is 57 cylinders. If the access queue for a particular device increases, the average number of cylinders to jump to service the next request decreases by taking the next request as the request on the queue that is nearest to the current position.

This scheduling process when combined with clustering and other placement strategies can considerably improve access times.

3.9.6. Adaptability of D.B.M.S.

The data base environment will constantly change, the D.B.M.S. must be able to adapt the data base to change within its environment.

Some of the changes that can occur:-

(1) At different times different user groupings dominate the use of the D.B.M.S. Each user will have his/her own individual requirements for accessing.

(2) Within the user group, there is a natural fluctuation of interest over a period of time, leading to different system demands.

(3) As updates to the data base occur, data elements change in time, leading to a deterioration in performance.

(4) As time passes, the design of the system becomes less and less relevant to the new environment. The system cannot be designed to predict changes in detail unless the D.B.M.S. is inherently adaptable.

In concluding the section on performance, 2 major questions are worth considering:-

(1) How can a system be designed to minimise the peripheral access contribution to the overall response time?

(2) Which designs and strategies permit a data base to age without permitting the performance to deteriorate?

3.10. REFERENCES.

(2) 'File Definition and Logical Data Independence'
C.J. Date and P. Hopewell.

(3) 'Further Normalisation of the Data Base Relational Model'
E.F. Codd.

(4) 'A Model for Access Control'
Peter S. Browne and Dennis D. Steinauer.

(5) Conference Paper on Integrity presented by Paul Peck at the 'Data Base Design and Implementation'
ONLINE Conference 1974.

(6) 'Adaptability to Change in Large Data Base Information Retrieval Systems'
C.J. Bell, B.K. Aldred and T.W. Rogers
I.B.M. UK Report UKSC-0027.

4. INFORMATION SYSTEMS.

4.1. OPERATION AND INFORMATION SYSTEMS.

Operation Systems - input to such systems have been anticipated and consequently included in the design.

Information Systems - input to this type of system is unanticipated and cannot be built into the design. The request can only be resolved when the request for information is made.

There are 4 categories of systems:-

(1) Scheduled production of predefined information. Typical batch systems with no on-line facilities.

(2) Scheduled production of undefined information.

(3) On demand production of predefined information. This is the normal on-line mode.

(4) On demand production of information defined when needed.

This last category is commonly called an INFORMATION SYSTEM.

Examples of Operation Systems are:-

- banking systems with tellers;
- airline reservation systems;
- sales order entry and enquiry systems.

Examples of information systems:-

- library information retrieval systems;
- marketing information systems;
- personnel information systems.

There are far more operation systems than information systems. The cost of operation systems are more tangible and are therefore more acceptable to management. The expertise in developing operation systems is much wider than expertise in information system development.

The data structures used in information systems are considerably different to those used in operation systems. Information system data is usually based on summary data. This fact often precipitates the decision to have two separate data bases containing in essence the same data, but with the information data base created and updated from the operation data base.

The reasons for this split are:-

(1) The information physical data structures are such that it is difficult and/or excessively time consuming to keep data up to date.

(2) It is difficult without a lengthy operation (best done off-line) to insert or delete the information data because of its summary nature. For operation systems where the data is quite basic, this is usually quite easy.

(3) Operation systems because they deal with the operation of the organisation must be up to date, usually an information data base can be up to 24 hours or more out of date.

(4) The operation systems handle high transaction volumes needing quick access. Information system queries are small in number but tend to be long in response.

(5) The information system tends to access summary information without all the detail used in the operation system.

(6) Operation systems must usually be updated on-line while information systems can usually be updated off-line.

Operation systems are usually installed before information systems. One of the biggest problems in building an information system is to define what the system should provide. It is true to say that information systems although appearing to have the potential to provide great benefits to an organisation, are expensive and lacking either flexibility or responsiveness.

4.2. CHARACTERISTICS OF INFORMATION SYSTEMS.

4.2.1. Levels of Information and Operation.

Many people consider data base systems are synonymous with information systems. To build an information system it is desirable if not essential to use a data base system.

The operations, that are the basis of the running of an organisation and which interact with the data base system, can be grouped into 3 levels:-

(1) Level 1 operations can be almost completely automated.

(2) Level 2 operations can be partially automated but require management involvement.

(3) Level 3 operations require intelligent human thinking with assistance from computers.

Level 1 (Routine operations and reflex actions).

- recording orders;
- shop floor data collection;
- bill-of-materials;
- inventory;
- re-ordering;
- goods received;
- general ledger;
- invoicing;
- budget accounting;
- costing;
- payroll.

Level 2 (well defined management control).

- setting working budgets;
- choosing suppliers;
- sales management;
- short term forecasts;
- production scheduling;
- formulation of rules for routine operations.

Level 3 (strategic management planning).

- determining markets;
- long range forecasting;
- direct research;
- setting financial policies.

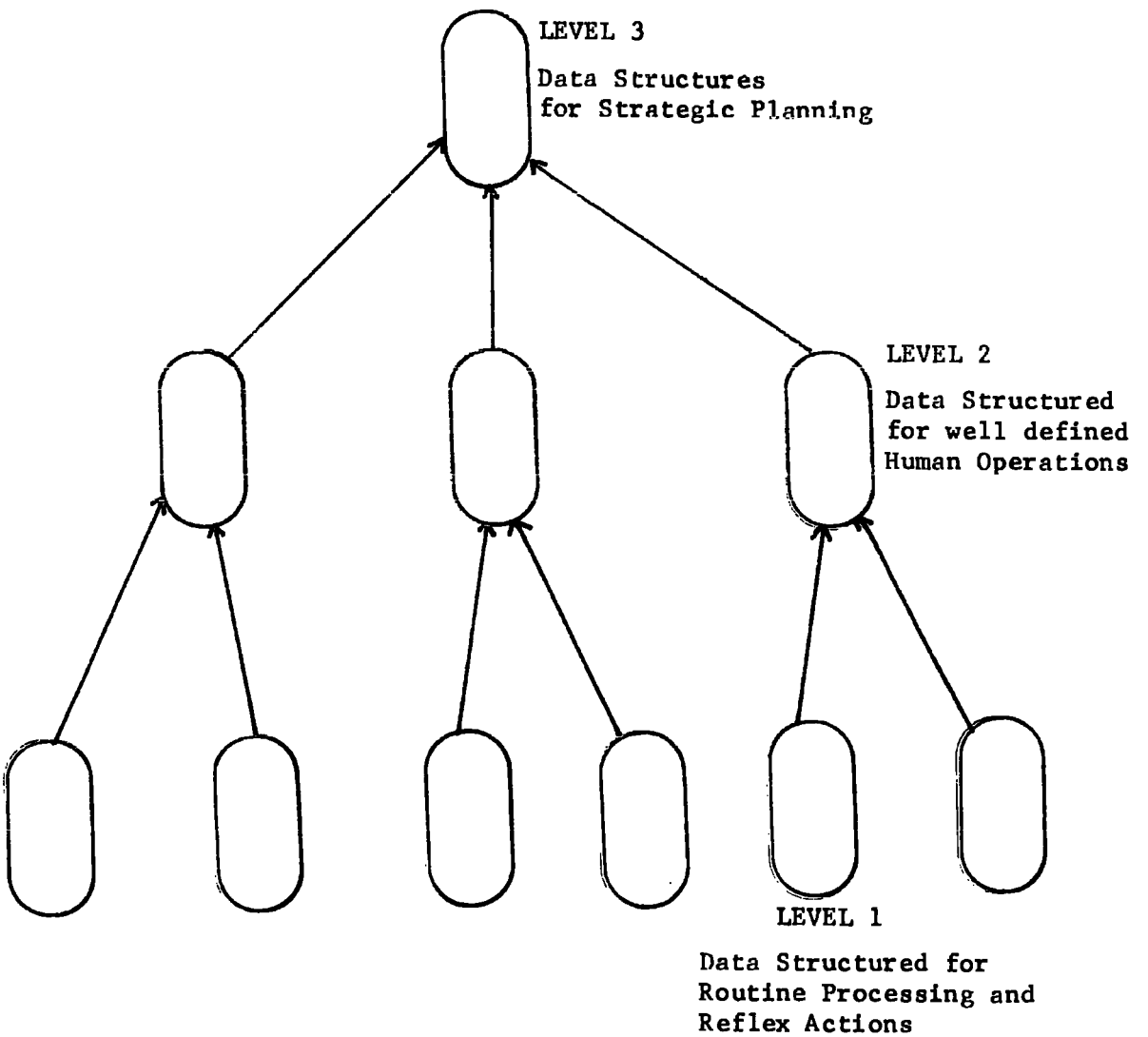


FIGURE 4.1. DATA STRUCTURE LEVELS.

It is desirable that as many operations as possible are brought into level 1 and that as much assistance as possible is given to level 3.

The data for the 3 levels is likely to differ in structure. A tight operational structure is necessary for level 1 also the data for level 1 can be in many distributed physical locations. Some of level 1 will also be in level 2. A further summary of level 2 will appear in level 3. It is not possible to permit management to 'fish' in a reservoir of operational raw data. The data must be processed in some way to present summary information.

At the moment, it is better to write special one-off programs for top managers because generally they do not use a terminal facility effectively. Lower levels of management use terminals more effectively because information requirements are usually simpler and the requirements tend to be anticipated.

Certain comments can be made concerning information systems for top management:-

(1) Staff providing information for top management should have access to different data bases. Terminals should be installed for this purpose in head office location.

(2) The head office should be staffed by information specialists who know the nature of the various data base systems and understand how to extract information from them. A high level information system needs the intelligence of a skilled librarian.

(3) Terminal dialogue should be similar in structure across all information systems so that one person can become expert in using all of them.

(4) The various levels of data base should be constructed using the same data description language.

(5) D.B.M.S. should have a data base interrogation language of a comprehensive nature, so that unanticipated complex requests can be dealt with fairly quickly.

4.2.2. Information Request Probabilities.

Information system needs can be graded on the basis of how predictable are the requests. The degree of predictability has a major effect on the design of such systems.

At the bottom of the scale there is information that is requested every day.

Information that is not requested so repetitively may also be requested in a different manner as a slight variation. Because the information is asked for less frequently, the answer must be more valuable to justify the cost.

Often the most valuable information stored is not requested at all.

It is much easier if requests for information are precise. This can be achieved by a precise interrogation language being used by an information specialist working on behalf of top management.

The information system should provide pre-search statistics to give information on the overhead of answering the request. This in itself improves the precision of the request. In many systems the directories that enable the requests to be answered are larger than the data files themselves.

The hallmark of good information searching is asking the right questions by constantly varying the questioning, searching for the answer which is of most use.

In the use of an information system there is no substitute for human intelligence, using the machine alone is not sufficient, the human specialist is required who understands the information.

4.3. INFORMATION SYSTEMS DESIGN.

4.3.1. Types of Data System.

(1) The simple data processing system has the following characteristics:-

(a) The system serves the clerk not the executive.

(b) The system does not generate information needed by management.

(c) The system permits data interpretation at such a low level that distortion is possible.

(d) Information that finally reaches top management from this type of system has been filtered by several people according to their concept of management needs and not according to a rigorous statement of management's needs.

This is adequate to support the clerical function but to support management decisions it is frustrating and inadequate.

The simple data processing system has a number of advantages:-

(a) A low implementation cost. Each task is small and easily understood by the user and systems analyst. The tasks in the system are easy to design, program and test.

(b) A low operating cost, the smallest systems on the smallest machines.

(c) Easy estimating.

(d) Political considerations are minimal. All tasks are minimal. All tasks are independent, each serves only one master.

The main disadvantage is that such systems cannot grow into an integrated system, a new design is required. The system sees only individual data elements and not the interaction between elements, the structure needed to tie tasks together is lacking. Integrated file systems can answer such problems.

(2) The second level of system, the integrated file system, can support the answering of random queries. Such systems are based on the availability of structured data files. The major problem in building an integrated data system is the cost, which is due to the need to thoroughly understand all inter-relationships which exist in the organisation. This

takes a great deal of time by all levels of personnel within a firm. If the business has vigorous growth and it is large the cost will not cancel the benefits.

(3) The definition of Management Information Systems (M.I.S.) is that they provide management with all the information needed to make the right decisions in running the affairs of the organisation.

The main difference between M.I.S. and an integrated file system is that the M.I.S. not only permits analysis of current and historic data but also supports simulation and prediction of the consequences of alternative courses of action. Integrated systems may provide reports on relationships of interest to management. The M.I.S. should go further to provide reports on relationships that management did not realise were significant. M.I.S. should be responsive to management's changing needs for information. The M.I.S. should direct output to the right level of the organisation and not bother upper management with data not relevant to its needs. The M.I.S. should keep track of trends in the data it is processing so that it can give warning when existing algorithms are becoming obsolete, when they should possibly be re-analysed and perhaps replaced.

4.3.2. Approaches to Information Systems Design.

There are 3 approaches to information systems design:-

(1) Set out the classic functions that should apply based on text book knowledge.

(2) To study the functions performed in the organisation and automate them.

(3) Study the information flow of the organisation and build the system that brings the basic information together and makes it accessible to all functional groups.

The first method is weak as it ignores the personality of the organisation and it is impossible to fit the organisation to the text book.

The second method is practical but it is also weak as it simply speeds up the way the business has always been run. If the business is not well suited to data processing techniques it will fail. Its only use is when the workload has outstripped the manual method.

The third method is easily the best. It takes advantage of the fact that the business organisation is creating the information that flows through it. The analyst can form an information model by determining what information is originated in each department, and what information is used in each department. If constructed properly, the model will describe the source and destination of all information, the elements of the information and their interaction.

4.3.3. Communication with Information Systems.

The very popular commercial programming language COBOL has never been popular in communicating with information systems. The reasons are:-

(1) The capability of the language has lagged behind user requirements.

(2) The amount of programming required to build and to maintain files is extensive.

(3) Languageslike COBOL are programmer's languages, lacking the naturalness desired by users.

The ideal language from the user's point of view is free form English, but in general the more free form the language the less precise is the language. Usually languages are used that have keywords or formatted lists of arguments. It is also probable that the language will give the capability to use Boolean logic.

4.3.4. Problems with Information Systems in On-line Mode.

There is no way of knowing from a query input how much information will be obtained in the answer. The system should provide statistics as an estimate of the response. The system should give the user the

option to continue or to have results printed out or possibly to cut down the information in the response. There is a tendency for users used to batch systems with enormous output reports to consider a computer as a sophisticated accounting machine. There is then an education programme required to teach the use of information systems and how to best obtain information from the data base. It is also possible to use graphics terminals to obtain graphic information.

Information systems can be simple or complex depending on:-

(1) Size or complexity of data base.

(2) The frequency and complexity of queries.

(3) The number and complexity of algorithms applied to extract data from the data base.

(4) The size and complexity of output reports.

(5) The organisational level and the authority of users.

Complex systems are the province of the information specialist. He should be a person trained to be the manager's agent in dealing with the data base. He can be trained to learn and use the system characteristics. The manager cannot be expected to become proficient in the use of the system, he is the decision maker.

4.4. MANAGEMENT INFORMATION SYSTEMS (M.I.S.)

The decisions made by managers are often under conditions of uncertainty. Whenever a situation recurs often enough so that a procedure can be developed to handle it, it can be relegated to routine, no longer requiring management decision. The manager is always dealing with problems that are new.

The proper way to develop an information system, is to build it in increments. The first phase of the M.I.S. is to integrate the experience of the organisation with the manager's requests for information.

4.4.1. Management Feedback.

The evalution of how well an M.I.S. is working as an aid to management is based on management feedback. The result of the evaluation is the setting of objectives for the next M.I.S. phase of development. The new objectives omit functions that generate routine information, this is passed to administrative data processing. New capabilities are identified and added by the analyst and the cycle continues. Each phase comes closer to satisfying the current needs of management because:-

(1) The interaction between manager and system is more 'comfortable'.

(2) The information output is more relevant.

For such a feedback system, the programs must be modular to facilitate regular alteration. Managers remain the decision makers, in fact, he becomes more important because he is freed from trivia.

4.4.2. Objectives of Management Information Systems.

Important M.I.S. considerations:-

(1) Is the system structured so that additional applications can be added? It is best if the M.I.S. starts small and progresses slowly.

(2) Applications must be adaptable to the user environment.

(3) Application routines should be modular so that they can be considered a processing entity.

(4) Output formatting should be independent of the application programs.

4.4.3. Future Management Information Systems.

The future M.I.S. will consist of 4 distinct parts:-

(1) Observation - performed by the data base management system;

(2) Inference - M.I.S.;

(3) Evaluation - M.I.S.;

(4) Decision - Manager.

5. THE DATA BASE ENVIRONMENT.

To draw together the subjects of the preceding chapters it is necessary to talk about the organisational environment which changes with the introduction and evolution of data base management.

The single most important person in the new data base environment is a person called the DATA BASE ADMINISTRATOR (D.B.A.). It is his job to control and administer the data base and advise on management policy towards data. He has a number of tools at his disposal, notably a source of information called the DATA DICTIONARY.

The user's involvement and commitment to data base implementation is the first and most important step to a successful implementation. The user's needs must be fulfilled. The user will only find the accessing of the data base acceptable if it is easy for him to obtain the information. The terminal facilities, including an easy and effective terminal dialogue, are vital.

Planning, implementation and future policy in the evolving data base world will mean that the data resource and information reservoir is put to the best use as quickly and as cost effectively as possible.

The data base environment will probably be unrecognisable in 10 years time with the development of technology in this field. There will probably be more advancement in technology in data base management in this period than in any other area of information technology.

5.1. THE DATA BASE ADMINISTRATOR.

5.1.1. The Administrator's Role.

One of the first considerations is where within the organisation should this function be placed? In many organisations it would seem the higher the function is placed the better the results obtained. The danger is that if the position is too low then the person is always involved with technicalities, and the person cannot be in a position to take a corporate view of the role of data. Too many organisations have permitted application-level data bases to be developed without co-ordination or central management.

The position first appeared in the early 1970's. It covered many functions from a support role similar to a systems programmer to the overall custodian of the corporate data resource, external to the data processing department.

The administrator's function is also dependent on the type of information systems employed and the role that the D.B.M.S. fulfils:-

(1) The D.B.M.S. is used for one particular system which has a complex data structuring requirement.

(2) Only a low level of data integration is employed e.g. production control, bill of materials, order processing etc.

(3) The ultimate in cost and desirability, the corporate integration of data which can support corporate information management systems. Such systems cut across functional operating lines. Even at present there are only 1 or 2 examples of such systems.

Each different type of system needs a different type of administrator, with different authorities and different responsibilities.

It has been said that the work is 60% non-technical and 40% technical. The non-technical expertise must include an ability to understand the business, the way it operates, the business data and how to model that data. Nevertheless the person must be technically very able, having the ability to sell his ideas, to communicate orally and by writing, to be a diplomat. Selling the idea of data base management at the bottom does not work. Selling at the top and permitting the enthusiasm to filter down to people at lower levels is the only way, but it takes a long time. The person must impress upon departments that although they are the creators and thereby the owners of data, that data must be shared with other departments for the benefit of the organisation.

Usually the position does commence as a technical position but it will evolve, functionally and organisationally into a management position. The question sometimes arises, should it or should it not be a part of data processing? The data base administrator must be able to resource allocate and to resolve conflicts.

The administrator's function, having evolved, should consist of 2 groups:-

(1) A technical group concerned with the physical aspects of the data base.

(2) A group dealing with logical aspects and with the business function.

The need to have a data base administrator does not necessarily arise from the use of a D.B.M.S. The function arises mainly from the concept of integrating data, with systems and programming becoming data oriented rather than process oriented as they had been in the past, with data on files satisfying a single program requirement. The administrator then becomes the single control point of data, transcending departmental boundaries.

5.1.2. The Administrator's Qualifications.

The following qualifications are necessary for the position:-

(1) The person should have a complete knowledge of the business to enable him to model the natural data relationships of the business. This knowledge is needed to guide the data base development along an orderly and evolutionary path.

(2) He/she should have more knowledge than anyone else in the organisation with regard to data base structure.

(3) The person must be familiar with all applications.

(4) The administrator must be diplomatic, a good negotiator, because usually users are resistant to change. To tell the user that his system is going to cost more to run to support an improved information service which is of benefit to the organisation as a whole, is not easy. The administrator must be able to convince users of the utility of the data base approach.

(5) He must be technically skilled with an in-depth knowledge of the current 'state of the art' of data base technology and very deep knowledge of the D.B.M.S. (if one is being used).

5.1.3. The Aministrator's Basic Tool - The Data Dictionary.

The administrator must know how, when and where data is used. There are 2 methods at his disposal:-

(1) A Data Element Dictionary.
This file describes all the data elements in use in the organisation. It includes all the elements' characteristics, properties and relationships.

(2) The administrator must be able to gather information on the way that data in the data base is being used. The D.B.M.S. should supply the administrator with data gathering statistics utilities so that the person can determine data element utilisation.

The data dictionary is used by other people. The systems analyst needs to find what data is available in designing a new application. Programmers need to know that they have the correct name and coding for the data elements in their programs. Terminal users require guidance when they interrogate the data base. Management can find out what data can be made available for them.

If the dictionary is integrated into the D.B.M.S. structure, it may be used by the software to maintain data descriptions and to compile programs which reference data elements whose descriptors are in the data dictionary.

A data dictionary is a data base describing the organisation's data bases, it is the repository of data about data.

The dictionary data base will include the following data types:-

- data item;
- data aggregate;
- sub-schema record;
- schema record;
- sub-schema;
- schema;
- physical data base;
- system;
- source of data;
- document or dialogue in which data is used (data output);
- program using the data;
- user department or person owning the data.

5.1.4. Data Standardisation.

If the programmer wants a data element amendment he must contact the data base administrator who must study the proposed amendment to ensure there is no duplication or conflict with existing data. If the change is acceptable the administrator must check if other users are affected. If it does not affect other users the administrator enters the change in the data dictionary as 'concurred'. If other users are affected, the data amendment is passed to other users for their agreement and the data element is set in the dictionary as 'proposed'.

If there is disagreement between users, the administrator must act as mediator and resolve the problem.

When a 'concurred' program is used to test the data element alteration and if proven the alteration is entered in the dictionary as 'approved'.

By using a centralised data dictionary, programs can be developed to use the same data for several different locations (divisions) in the company.

The benefits of such a dictionary are:-

(1) To enable element standardisation.

(2) As a central reference source for data names and definitions.

There are problems if a dictionary is not used:-

(1) There develops communication difficulties between user and analyst/ programming staff, misunderstandings develop.

(2) Personnel turnover requires a long learning process because new people are forced to look at code to understand the data used.

(3) Duplicate data maintenance and storage requirements are increased because of the large number of data elements with non-descriptive names.

(4) The lack of data item standardisation makes system testing difficult.

The data dictionary develops as the systems develop.

The types of information stored for each data element are:-

- what is its description?
- what is its location?
- how is it used?
- how does it relate to other data elements?
- who uses it?
- where is it used? (application program, report)
- how often is it used?

The benefits of using a dictionary are:-

- faster debugging and testing of programs and systems;
- easier future integration;
- reliable and up-to-date information about the data base;
- standardisation achieved on usage, labels and definitions;
- aids data base design and planning;
- fixes the responsibility for data definition;
- communicates changes to the user.

5.1.5. Dictionary Contents.

The data item has 14 characteristics:-

- name;
- definition;
- size;
- character type;
- coding;
- information sensitivity;
- data item source;
- data item users;
- functions;
- where data item is employed;
- set of acceptable values;
- data chain;
- projected users;
- controlling authority.

Each of the above must be determined before the item is entered in dictionary. Each data item is cross-referenced to programs, records, input, output and messages. The dictionary must be easy to use. All programmers when developing their programming specifications must use the dictionary.

A complete system is needed to maintain, update and use the dictionary. If programmers think that the dictionary is hindering development it will not be used. The administrator must ensure that the dictionary is easy to use and is updated timely. In this way the dictionary can be used to find out which programs are affected by an alteration of data by maintaining a list of data items used by a program and what types of access they have for each data item.

5.1.6. Data Dictionaries and Directories.

The dictionary is the information on data that is required for human understanding of the data.

The data directory is the information on the data that is important for the mechanised processing of data.

It is a commonly held view that the dictionary/directory files should drive the D.B.M.S. The dictionary system would in this case be a very complex piece of software.

5.1.7. The Administrator's Functions.

The functions of the administrator are many and varied, from the administrative to the technical.

The person must maintain data definitions and the schema and he/she must resolve any departmental arguments concerning the sharing of data and any data structuring or data definition conflicts. Department managers tend to be reluctant to release their control over data. The use of the data base concept suggests the integration of systems which involves the restructuring of earlier files and the re-writing of programs, the cost of which departments will argue against.

The administrator also provides facilities for the application programmers and systems analysts to permit them to write programs as easily as possible to make accessing the data base easy and to hide much of the data management problems from the application staff.

The administrator is constantly involved with performance, i.e. the 'tuning' of physical data structures to improve performance. The data base usage must be constantly monitored by the administrator so that performance problems may be anticipated.

He/she must also provide consultative and counselling services to users in 3 categories:-

(1) The application programmer is helped with the data definition.

(2) The systems analyst is helped to understand the data base structure.

(3) The user is helped to understand what data is available to him.

To summarise the functions that the administrator may be involved in during design and implementation:-

(1) The designing of data structures to model business structures and to satisfy data requirement of data base users.

(2) The selecting, extending or writing of a D.B.M.S. to support these data structures.

(3) The naming of entities of data involved in the data structure.

(4) The encoding of data structure descriptions using D.B.M.S. facilities.

(5) The selection of data ordering and searching strategies.

(6) Selecting and defining methods of:-
- capturing data;
- loading the data base;
- converting existing files into the data base;
- the restructuring of the data base.

(7) Designing privacy procedures using those provided by the D.B.M.S.

(8) Controlling (thereby minimising) the re-writing of application programs.

(9) The maintenance of a news service so that new application programs can use existing data.

(10) Extending the scope of application of the data base.

(11) Designing mechanisms to ensure the accuracy, security and the continued availability of data in the data base.

(12) Designing mechanisms for the controlling of any archival storage.

(13) The design of procedures for the manual intervention by the computer's operating staff.

(14) The allocation of data on storage devices in order to optimise performance.

(15) Predicting the requirements for any new hardware, storage media or software.

(16) Providing freedom from device dependence and data structure dependence for all application programs.

(17) Documenting the system.

The administrator must perform certain operational control tasks:-

(18) Monitoring data base performance, the frequency of use of the data base made by the users and the response times being obtained.

(19) To check whether there are any attempted breaches of the privacy system.

(20) To ensure that data validity checks are not by-passed and that irrelevant data is removed from the data base.

(21) To check that application programs comply with procedures and self-imposed discipline. This is necessary if there are many archiving and error recovery techniques used.

(22) The imposition of adequate testing for new application programs in a secure environment before allowing them to act on an operational data base.

(23) The changing of user priorities.

(24) The changing of privacy keys if required.

(25) To ensure that data updates are prompt and accurate.

(26) The recording, documenting and reporting of data base performance.

5.1.8. The Administrator and Data Base Performance.

The points that may concern the administrator with regard to data base performance are as follows:-

(1) A study of the reponse times, throughout, system utilisation and efficiency of programs.

(2) Checking for performance deterioration and reviewing utilisation reports to determine if more direct access storage is required or the data base needs reorganising.

(3) The administrator should have simulators available to him, to enable the D.B.A. to study effects of hardware and software.

(4) Hardware monitoring is useful in the determining of hardware utilisation and to highlight disc or channel contention problems. A software monitor would be useful in determining operating system overhead and the efficiency of application programs.

The D.B.A. must be able to improve performance by:-

(1) changing the physical data base structure;

(2) changing the buffer sizes;

(3) devising recovery procedures.

There are several different trade off's in performance which the D.B.A. must bare in mind:-

(1) device utilisation and throughput against turnaround time;

(2) system development cost against system maintenance cost;

(3) program development cost against run time efficiency;

(4) people time against computer time;

(5) main and secondary storage space against computer time;

(6) secondary storage space utilisation against main storage space utilisation;

(7) ease of insertion, against ease of maintenance, against ease of deletion, against ease of search and retrieval;

(8) flexibility against complexity;

(9) redundancy of data against redundancy of directories;

(10) overheads for integrity, security and privacy.

5.1.9. The Administrator and Data Base Integrity.

The D.B.A. needs to set out rules to control integrity. The administrator must be able to access the data base at a different integrity level than normal users.

The D.B.A. must be able to use:-

(1) Control parameters for automatic selection process.

(2) Override facilities for these processes.

(3) The administrator must define automatic decision processes and the logical steps presumed to take place in those processes that are not automatic.

(4) A means of registering the results of manual decisions in terms of the use of particular integrity control tools.

Such automatic decision processes may take place in:-

(1) any D.D.L./D.M.L./D.S.L. processors;

(2) the run-time D.B.M.S. software;

(3) D.B.M.S. utilities.

The D.B.A. must monitor data base integrity with particular reference to:-

(1) Ensure that planned duplication of data is kept consistent by the data base system and that users keep the data base accurate and timely at all times.

(2) Determine the implications of the number of users on integrity and privacy.

(3) Ensure all input data is validated.

(4) To ensure that there is proper control over the program interface to the data base.

(5) The examination of the data base to ensure the purity of data and to determine what data can be deleted by whom.

(6) He must be responsible for quality assurance checks.

5.1.10. The Administrator and D.B.M.S. Failure.

The D.B.A. is responsible for decisions arrived at in the event of data base failure. The decisions concern:-

(1) Recovery procedures for each fail type.

(2) Exception Rules.

(3) Contingency processing with a hierarchy of recovery techniques to be attempted if a given technique fails.

(4) Communication from the D.B.M.S. to the D.B.A. when the D.B.M.S. requests manual intervention.

(5) How does the D.B.M.S. recognise exception conditions which it cannot handle automatically and for which the administrator must be informed?

(6) The operating procedures associated with recovery.

The D.B.A. requires a system control language for use at the time of failure to enable manual selection and direct control of recovery processes. The D.B.A. and D.B.M.S. between them must make the following decisions when a failure occurs:-

(1) Is the damage global or localised?
(2) Which run-units can be restarted automatically?
(3) Which run-units must be restarted manually?
(4) Which recovery files are available?
(5) How should recovery be initiated?
(6) Which users should be notified?

5.1.11. Installation Definition.

The administrator should be provided with facilities to enable the D.B.M.S. to be tailored to the installation's requirements. The following areas of definition are important:-

(1) Facilities are required to enable the D.B.A. to specify maximum levels for recovery performance for each class of failure.

(2) The D.B.M.S. should provide attachment points with standardised interfaces for addition or enhancement of facilities by the D.B.A.

(3) The definition of 'outer limits' of availability and recovery requirements.

(4) Is the on-line processing updating or read-only?

(5) Do on-line run-units have 1 or more success-units?

(6) What levels of concurrency are chosen for performance?

(7) What levels of concurrency are chosen by circumstance?

(8) What types of run-unit are involved in concurrency?

5.1.12. The Administrator's Influence on Application Development.

The administrator must provide compatability so that non-data base files can be supported outside the D.B.M.S. environment so that old systems can use data stored in the data base in the old format before D.B.M.S.

A test data base must be created and maintained to enable changes to D.B.M.S. software. Similarly testing facilities must be provided for newly developed application systems.

The D.B.A. must take an active role in application development. The person must identify data base requirements for future applications and schedule the required data base implementation. The systems analyst reviews with the administrator his data requirements. As a result, the administrator must look at the data dictionary to determine whether the data required is already on the data base.

5.1.13. The Administrator and User Interface.

The administrator must ensure that there is adequate education and documentation. Users must be educated in the principles and policies governing data base design and usage. Users must have details of what the data base contains and how it can be used. The application programmer must have access to the dictionary and access to the cross-reference relationships between data elements and programs. The programmer must be aware of the rules and instructions on how to access the data base.

The relationship the D.B.A. has with users is very important. The maintenance of good communication with all levels of user to provide advice and guidance, to promote effective use of data base and data base software. The D.B.A. must take pains to explain data base benefits for users. To that end the administrator must be able to argue articulately to ensure the right course of action. The D.B.A. must be able to turn down a request from a programmer or systems analyst to use data, the person must fully understand the user's needs and be able to explain why he/she is not permitting data to be used. The administrator must insist on a long term plan for data base development be drawn up and that short term objectives will not impede future developments.

5.2. THE USER.

When referring to the USER in this section the type of user that is meant is the person in departments other than management services who will only use a data processing service if it is of practical benefit to his own sphere of work. It is this person who must be convinced of the usefulness of the data base environment, if the data base is not to become a white elephant. The most important interface of such a user is the visual display unit.

5.2.1. Mode of Operation.

There are 2 possible modes:-

(1) The user is in control, conveying questions/instructions to a language interface.

(2) The system is in control, presenting the user with a menu-like selection or series of questions to elicit the necessary information from the user.

The user should be able to alternate between either modes.

5.2.2. The User/System Interface.

No special control language should be required. If the user is forced to obtain information in an artificial way, the person will gradually cease using the facility. The user language must be adaptable to a large variety of input/output devices in a variety of environments, e.g. the office and the shop floor. The conversational software should be helpful to the user, validating requests, prompting the operator, attempting to judge what the operator intends and to suggest what the user might try next.

The user should be able to store and catalogue a request and use it repeatedly. When repeating the request the user should have the facility to change only those parts that need to change. Consequently a hierarchy of defaults are required:-

(1) Functional defaults provided by the vendor.
(2) Installation defaults reflecting the corporate rules and standards.
(3) User defaults defined by the administrator, reflecting defaults unique to the particular class of user.
(4) User defaults defined by the user himself for his own convenience.

5.2.3. The User Profile.

The D.B.A. must have a profile of each user in order to protect the integrity of the data base.

There are 3 categories of profile information:-

(1) Information restricting the user by rules established by the organisation.

(2) Information provided to resolve situations the user is not capable of resolving.

(3) Information stored by the user for his own convenience to aid in the use of the language.

The last category is set up and maintained by the user, the others are maintained by the D.B.A.

The types of information in the profile are:-

- Data elements users can retrieve.
- Data elements users can update.
- The limit of machine resources the user can monopolise.
- The 'world' in which the user normally operates.
- The aliases for data names and functions which are unique to a given user or class of user.
- The language functions the user may or may not use.
- The output devices the person can use.
- The devices where information is to be sent if not explicitly defined by him.
- Defaults set up by the user.

5.2.4. The Query Language.

The terminal query language must try to achieve ease and flexibility with effectiveness of use.

The problem with natural language is in the significance of key words. If the language does not understand certain words i.e. it does not possess a basic vocabulary, this can lead to user frustration at a lack of system responsiveness. Most importantly the user should be guided through the interrogation process.

The best man-machine dialogue is where precise and highly formatted information is provided to the automated system by the user in relatively

small amounts, so that the system returns with statistics relevant to the impending search or with information that will help formulate the query further.

There are 2 types of processing:-

(1) Intra-record processing where the results of queries and consequently reply is embodied within a single identifiable record.

(2) Inter-record processing where an intermediate file is generated with data obtained from several records which may be sorted to produce reports etc.

Information requests may not require any interaction between man and machine and may be serviced when scheduled by the machine; these are often referred to as 'off-line' queries. Such a facility may be entirely sufficient for the users needs. The advantages of 'on-line' facilities are not only the probability of a quicker response but also the capability for the user to narrow his search to obtain the right amount of data. Certainly off-line queries tend to produce superfluous information. This 'homing-in' to data the user requires is called heuristic searching.

The facilities that such a language should provide are:-

- the ability to define data structures;
- to define transactions for updating files;
- the definition of logical and arithmetic operations;
- to define in detail the layout of reports to be generated;
- to define tables to be used;
- to catalogue processing requests;
- data manipulation facilities should be provided to count, sort or total items;
- to provide report formatting facilities;
- the ability to sequence data;
- computational facilities involving addition, subtraction, multiplication and division;
- the ability to create files for future use.

There are several functional capabilities that a language such as this should possess.

Input.

The language should validate data input by rules set in the data dictionary. The types of checks that would be required:-

- class checks;
- range checks;
- specific value checks;
- field inter-relationability checks.

Retrieval.

Certain advanced facilities are required:-

- tutorial functions to guide users through searches in a step-by-step learning process;
- the ability to select a subset of the data base by statistical sampling;
- to specify the sequence of data presented to the user.

Data Manipulation.

The different types of function that should be provided:-

- arithmetric computation;
- business functions including accumulation, percentages, mean, standard deviation, maximum and minimum, compound interest, depreciation;
- maths functions including sine, tangent and absolute value;
- array operations;
- data string processing including locate, insert, delete, extract, order and align.

Further data manipulation facilities that should be provided are:-

- the automatic handling of scaling of items.
- the automatic handling of the conversion of data representations;
- the preserving of significance;
- the automatic conversion from one unit to another;
- automatic decimal alignment and rounding;
- truncation and alphabetic padding.

Updating.

Changes can be made to the data base in 2 ways:-

(1) modifying the contents of existing data items;

(2) adding or deleting items.

It must be possible to have mass updating and unit changes. A trivial update could trigger a large chain of updates. The integrity of the data base must be maintained even to the extent of preventing the user from the initial update.

It should be possible to perform pseudo-updates i.e. to ask what are the effects if an update is done without actually doing the update. The facility should be provided to back out of updates from a certain point in time. This facility should be provided to both the users and the D.B.A.

Output.

Several types of output should be supported such as graphic terminals.

New Functions.

The interrogation system should be open-ended to permit users to generate new functions.

Interrogation should have restrictions. If, for instance, a query is long or complex, the interrogation facility should return an indication, by statistics, of the amount of processing involved (and/or output produced) from the enquiry. Such a set of statistics are called pre-search statistics. If the user confirms the enquiry, then and only then will the search for the information take place.

This is because some searches which may seem quite small and simple may take a lot of resources. If the search should appear large it may be postponed until a slack period.

5.2.5. Man-Machine Psychology.

Divising dialogues between man and machine is a new form of literacy. Users should not need to remember mnemonics. Responses should be clear and concise. The data base user should also be aware of what data is available to him with security controls restricting his view to that data he is permitted to view. The user should be able to obtain the definitions and other information about the data items being used.

The man-machine dialogue is 2-way, the better solution is not always obtained if the conversation is driven by man. If the computer initiates the exchange it is called a COMPUTER-INITIATED dialogue, otherwise the dialogue is called OPERATOR-INITIATED. Usually the former dialogue necessitates far more characters to be sent to the terminal. The dialogue for an operator who has learned a programming-like language is usually operator-initiated.

5.2.6. Types of Operator.

The Creative Operator.

Certain advanced facilities are required for creative, advanced terminal operation. Typically:-

(1) A fast terminal response is essential.

(2) Immediate availability of tools and techniques e.g. dictionary look-up, text editing, simulation.

(3) 3-dimensional graphics. A good example would be the need for animation in medicine.

The Totally Naive Operator.

In this type of operation, completely computer driven questioning is required. The operator can only answer 'YES', 'NO' or '?'. It may also be possible to use menu selection. Whatever techniques are used must assume a lack of training and shortage of intelligence.

The Untrained Operator.

It should be easy for an untrained operator to identify his/her self at a terminal and to be ready to open the dialogue. The terminal in giving the reply should be clear and should not leave the operator confused. The computer dialogue must clarify errors and unusual circumstances when they occur.

Graphic Terminal Operation.

This is a powerful information technique because a picture is worth a thousand words, it will become an 'indispensable tool' for managers and their staff.

Terminals for Management.

The essential property of such a dialogue is that it should not be difficult to use. The characteristics of such a user are:-

(1) A high intelligence.

(2) The manager requires a high 'information bandwidth' in the dialogue.

(3) A manager is too busy to have a training course.

(4) The manager will not remember mnemonics.

(5) He/she may be a highly impatient person.

(6) He/she will be dissatisfied with confused dialogue or unintelligent error messages.

There are 3 possible options to provide management with terminal facilities.

(1) To design a dialogue structure that is suitably simple.

(2) Provide specialist assistance to management.

(3) If the management information requirement is great an information room may be set up to which the manager can telephone his information request.

5.2.7. Response Times.

The response time of a system is the time interval between an event and the system's response to that event. In many cases a fast response may not be necessary for the efficient functioning of the system, indeed too fast a response may coerce a slow operator into attempting to keep up to the level of response of the machine, causing input errors.

Several different functions at the same terminal will probably require differing response times. Sometimes it will not matter if the information is 5 minutes, an hour, or a day out of date. Nevertheless if a user inputs something at a terminal he should receive a confirmation within seconds but the request may be queued ready for later execution. Response time to the same function should not vary very much. As a rule of the thumb, the standard deviation of response time should be less or equal to 50% of the mean response time.

The reasons that can be given for a fast response time are:-

(1) The need to meet response times to control machinery in process control or military applications.

(2) A fast response time may give economic savings that can be calculated in a tangible manner.

(3) For behavioural psychology, a fast response is necessary for an acceptable and efficient man-machine interaction in certain situations.

(4) A fast response time is impressive providing good public relations.

(5) Simply to 'keep up with the Jones's'.

There are economic savings to be had from a fast response. Longer response times can lead to a backlog of work. It may be possible to place an economic value on the delay caused to the person awaiting the response.

Psychological reaction to response times can be crucial to the success of a system. When a user expects a response in a certain time and that time is exceeded the time becomes an unnatural break. Complex enquiries may be made up of several enquiries each one continuing on a complex pattern of thinking. If the response time is greater than expected, the continuity of thinking may not be maintained because the operator may not be able to keep sufficient information in the human short-term memory.

There is not a straight line decrease in operator efficiency as response times increase. There is a sudden drop in mental efficiency when delays exceed a given point.

It is not necessary to have the same response time for every operation, for example while the human operator is following his/her train of thought, it is important to have a quick response, but at 'end of transaction', when all semantic checking of the validity of the enquiry or update of the data base is performed, a longer response time would be permitted.

5.2.8. User Terminal Considerations.

The machine's limitations must be recognised by everyone, the human advantage is clearly flexibility of intelligence. The questions that can be asked are, what are the conversations like, long and complex, or, short and simple? As a result what are the training requirements?

Some of the important considerations:-

- Only a small amount of information should be displayed at one time with only one idea for each display, no lengthy verbiage and no unwanted information.

- Keep the operator's response short which is rather difficult for data entry systems.

- The computer should always respond to the operator.

- Clarity should be the aim in the design of formats with guidelines laid down for display design. Different words and characters should be avoided if used by clerical operators. Instructions to operators should be concise and unambiguous. The display screen itself should be 'cleaned' wherever possible.

- It should be easy for the operator to ask for help and to back-track several displays to provide an easy means of correction.

Errors will often occur, some of which may not be the fault of the operator, more the result of bad design.

Operator Overload.

Presenting operators with too many messages or giving operators too many things to think about. Asking operators to make too complex a decision leads to errors and poor judgement.

Operator Boredom.

Making the operator's job too simple can lead to errors for intelligent operators. The design must strike a balance between making the operator's job too tedious and making it too complicated.

Lack of Motivation.

'Slave' operators need to be motivated, they should see the overall picture of what is happening and their part in it.

Handling Emergencies.

When failures occur, a set pattern for recovery must be spelt out to the operators and fully checked out for accuracy.

5.3. TESTING.

Testing in a data base environment is every bit as important and probably no less time consuming as in any other data processing environment. Probably 50% of time and effort in implementing a D.B.M.S. is in testing.

5.3.1. On-line System Testing Problems.

Problems are enlarged by the sophistication of processing, with:-

- complex networks;
- transaction input on a random basis;
- reliability requirements more stringent;
- the systems themselves becoming more complex.

The communication network is a potential source of error, it becomes more difficult to isolate and correct program errors.

On-line/real time software adds to the complexity of testing because of the complexities of message handling and queuing etc.

The reliability requirements become stiffer because if there is a batch failure, reports are late, but if there is an on-line failure it will not be accepted so easily by the user because he/she will not be able to fall back to manual procedures for long periods.

Systems become more complex because interface problems increase with several groups of people looking after different parts of the system. With the use of communciations there may be a hardware interface problem with different manufacturers hardware (lines, modems etc.).

5.3.2. Stages of On-line Data Base Testing.

There are 6 stages of testing data base applications. As the testing progresses through the stages, it becomes more difficult to find errors, consequently it is best to maximise completeness and comprehensiveness of the testing at the early stages of testing.

Stage 1 - Unit Testing.

Normally this stage is performed in single threading batch mode. The modules are tested singly, testing the main coding lines first, followed by the exception lines. No real data files are used.

The following conditions should be tested:-

- valid conditions;
- out of sequence conditions;
- out of limit conditions;
- each routine branched to as a result of major decisions;
- incomplete, invalid or missing input;
- numeric or alphabetic characters only;
- illogical conditions which violate logical relationships between fields;
- improper transaction codes.

Advantages of unit testing are:-

- Flexibility - a wide variety of input data conditions can be built into a test.
- Economy - network costs are eliminated.
- Minimisation of Error Sources - by eliminating the communications network a possible error source is being eliminated so that the detection and correction of problems are facilitated.

Over 70% of the errors should be discovered at this first stage of testing.

Stage 2 - Link Testing.

One of the most common sources of error is the interface between modules. These errors are often caused by differing interpretations of the interface often because two different modules are written by two different programmers. At this stage test data base files may well be used. The D.B.M.S. should provide extensive facilities for the setting up of test files.

Stage 3 - Multi-threading Testing.

This stage is used to detect errors associated with timing.

Stage 4 - Acceptance testing using the Communications Network.

This stage should be performed by data processing personnel other than the system developers. This test group is responsible for the final testing of the whole terminal, data base system including the checking of terminals, lines, modems, front end communications processors and all hardware and software interfaces.

The first part of the testing will involve only one terminal followed by multiple terminal multi-threading testing.

A load test is required to ensure the hardware and software will take the transaction volume expected in a 'live' situation.

A quality assurance test is required before the system is released for user acceptance testing.

The points that a quality assurance excercise should be looking at are:-

(1) To demonstrate that the system operates in accordance with user system specifications.

(2) To ensure that the system specification satisfies the user functional specification.

(3) To ensure that internal standards are adhered to.

Any deficiencies are taken back to the development group for correction and testing probably from stage 1 onwards, before another quality assurance review.

Stage 5 - User Acceptance Testing.

This is the last stage of testing before the system goes operational. The purpose is to satisfactorily process the real work load to check that the entire manual/machine/management system is satisfactory. If the system is replacing an existing system the old and new systems may be run in parallel to achieve a comparison.

System Cutover.

Once reliability, accuracy and adequate response times are demonstrated the cutover to actual operation can take place. Input to the old system (if there is one) should still be created for fallback purposes for a period. As quickly as possible data collection for the old system is stopped but with the new system still being closely monitored.

Stage 6 - Testing for On-going Development and Maintenance.

Errors will be found long after the system has been implemented. The capability to test both minor modifications and major enhancements must be built into the system. Such modifications will most certainly involve previous stages of testing.

There will be a need for a concurrent operational/test capability, this would be very difficult if all hardware is in use for 24 hours each day.

Major amendments must follow the whole cycle of testing, a 'System Change' Log and 'System Testing' Log should be maintained and in this way performance changes can be tied to system changes.

Stringent system documentation requirements must be set up. Documentation must be satisfactory before the system is changed.

5.4. DEVELOPING TECHNOLOGIES.

5.4.1. Distributed Data Bases.

Until the mid 1970's data files were processed by a single computer with the exception that certain data files would be transferred between machines on magnetic tape or disc cartridge. Even in such circumstances the machines would probably have to be made by the same manufacturer.

Since that time, the arguments for transferring data between machines by communications line and the advantages of holding data at differing locations have grown, although the complexity in software that this requires makes such a style of processing inadvisable at the moment.

Why should distributed data bases be such a desirability in the future?

Cost Reduction.

Storing data locally reduces data transmission costs, but economies of scale favour large centralised data storage; there is a trade off. The cost of small localised data storage is dropping more quickly than data transmission costs.

Load Considerations.

The data traffic load is greater than that which can be handled by a single data base manager. It is better to have several data base managers sharing the load with the capability to pass transactions between systems.

Localised Management.

Localised management and control of data has advantages. A local organisation has full responsibility for its data, for its accuracy and safe-keeping, although that data may still be accessible from other parts of the organisation.

Local data may still be required to conform to centrally agreed data definition, formats and schemas.

Mini-computers.

Locally stored data is more acceptable since the advent of mini-computers with data transmission capabilities.

There is a tendency for a distributed filing system to be used rather than a distributed data base system i.e. such filing systems tend to lack data independence.

Separate Operation Systems.

There may be several localised operation systems and one centralised information system, the latter having differently structured data bases designed for fast spontaneous searching.

In some cases the information system is based on a data base whereas the operation systems are based on file systems.

Man-Computer Dialogue.

The capability of having distributed terminal intelligence with localised data should improve the man-computer interface and provide greater efficiency.

Availability.

If data must have a high availability, it is sometimes good to store data in 2 separate locations in case of hardware failure.

Security.

There is more protection if data is split across several locations.

Terminal Versatility.

It becomes possible to give users a large range of data bases to choose from.

Computer Networks.

With the introduction of data communication networks, so the idea of a network of data bases gains acceptance.

There are different types of distributed system. The type of system chosen is dependent upon how the data base is being used. There are 4 types:-

(1) Data types and structures used at different locations are largely the same. The same schemas may be used for each of the separate data bases.

(2) Small peripheral storage units are employed to enhance the function of the central data base. The sub-schema used to describe the data held at the peripheral storage units may be derived from the schema used in the central system.

(3) Separate operation systems feed part of their data to a central information system. The operation systems have schemas in their own right and unlike type (2) above, each location could be a stand-alone system. Data types and formats used in the system are closely related and planned in an integrated fashion.

(4) The computers and the data bases in the network are entirely heterogeneous i.e. the data base are completely different logically based on computer hardware from different manufacturers.

The method of physically storing data can also vary. The network can be regarded as consisting of a number of nodes connected by communication lines. Each node consisting of one or more computers of varying sizes can be called an Information Processor. Each such processor may have individual, permanent files not to be accessed by any other information processor (I.P.). The data base files can be physically organised in different ways.

Individual Files.

Users create files at whichever I.P. is being used and in this sense the files are held geographically nearest to the source. The files may, however, be grouped in a location by type of file.

Duplicate Files.

The reasons for duplication are:-

- Increased access to a file by providing more paths to the data.
- Provide rapid back-up in the case of the failure of a device or an I.P.
- Decreased communications volume and/or decreased dependencies upon communications facilities between I.P's.

Splitting Files.

A logical file may be split across several I.P's. The splitting will usually be because of the accessing load which can provide the case for I.P's. to be not considered always as being geographically separate. Another reason for the splitting of a file may be on the basis of the origin of the data.

There are 2 major problems:-

(1) complex software is required;

(2) the complex software causes a processing overhead.

A full catalogue of the data base (dictionary/directory) is required at every I.P. The catalogue lists every file in the network, indicating where each sub-division of the file is located physically. Alternatively, a catalogue is kept at each I.P. for local data only and if necessary search for any other data across the I.P's.

To move data from one location to another:-

- transmit a copy of the whole file,

or

- the user program issues a request for data;
- the operating system discovers that the file is attached to another I.P.;
- the input/output request is sent to the appropriate I.P. using a high transmission priority;
- the other I.P. receives the request, processes it and returns either the requested data or a denial.

If 2 I.P's. are waiting for data from each other, a deadly embrace situation could occur.

It must be decided whether to transmit the transaction to the data or the data to the transaction, the decision will probably depend on the volume of data and how frequently transmission is required. It is undesirable to have more than one copy of the data being updated in different places at the same time.

Distributed Data Base Control.

Enough de-centralised control is required to give a local resolution to local problems and to encourage local initiative and innovation. Local problems are often well understood by locally working systems analysts. Sufficient centralised control is needed to pass data and programs from 1 location to another. New programs should be developed to be able to work with existing data from any locations.

Centralised Schemas.

It is possible to have no difference between schema and sub-schema for a distributed data base than for a centralised data base, but the physical structure/geographical location can alter without impacting the logical structure. The software to provide this physical data independence is complex.

There are 4 types of distributed data base organisation:-

(1) Sub-schemas and schemas are held separately at each data base location.

(2) The sub-schema and schema are held at one location and data is held at multiple locations.

(3) The sub-schema is localised while the schema is centralised.

(4) Separate localised sub-schemas and schemas at each location control data at different locations.

It is likely that at some future time, data base systems will come into existence providing logical, physical and possibly geographical data independence.

Data Location.

Locating data can be a simple or a complex procedure.

- The user may include the data location in the request but this means no geographical data independence.
- The user may specify information which can be used to deduce the data location.
- A user directory may be used to indicate where the data is located. Such a directory may be stored at one node in the network or at every node in the network. To store at every node takes extra space and may involve an update problem.

- Most data is required locally and it would be best to store such data locally. The local computer can keep a directory of data that it has, any other data request is passed to another machine at another node.

5.4.2. Operating Systems.

In the past operating systems and D.B.M.S. have developed independently.

The operating systems of the future will take into account the D.B.M.S. function and indeed the two functions will to some extent merge. This is already being seen to some degree with I.C.L's. Virtual Machine Environment's data management system. If such developments do take place with the development of hardware implementation for certain D.B.M.S. functions, the future of the stand-alone software D.B.M.S. package may well be in doubt.

5.4.3. Data Base Processors.

The tendency towards hardwar e implementation mentioned above is largely due to the move towards multi-processor architectures which consist of functionally discrete processors. The level of interface between such processors would probably be a macro string.

The advantages of such a 'back-end' processor type architecture are:-

- economy through specialisation;
- the enhanced ability to share data;
- improved data protection;
- an easier transfer of the data base to new computers;
- the user can have a unified data base system across different supplier's hardware;
- better integrity and security facilities.

5.4.4. Content Addressable Stores.

This new type of storage device with different properties may invalidate a great number of current assumptions underlying data base management ideas.

With current sequential processing hardware every record must be searched until the record with the required primary or secondary key is found. To alleviate the brute force of searching every record, pointers and index tables are introduced to direct the searching process. This stored linking of data is a storage overhead and may lead to inflexible data handling. It is the case that current D.B.M.S. all provide some type of stored data linkage and that all such linkage mechanisms provide either good updating or good retrieval accessing but not both. This means that data structuring for an operation system where on-line updating is important is totally different from the data structuring for an information system where the emphasis is on retrieval, even though the data used in both types of system may be the same. This trade-off situation between retrieval accessing characteristics and update accessing characteristics will disappear with the introduction of content addressable storage. All problems of this nature are overcome because searching is based on content rather than address. The storage mechanism consists of an associative memory which performs a parallel hardware search on the contents of the file against a bit mask to find the record with the required contents.

There are currently two major problems with associative memories:-

(1) The memory size is small by main memory standards.

(2) It is currently expensive due to the need for logic with every bit on which the search is made.

These disadvantages will disappear with the introduction of micro-processors and developments in electronic storage.

The use of micro-processors will take an important part in the development of D.B.M.S. Micro-processors will eventually be able to take over certain D.B.M.S. functions providing the basis for a mixed hardware/software system enabling the most flexibility and performance to be achieved together in the same system.

Currently associative memories are being used in the following ways:-

- Use small associative store to hold record identifiers and retrieval keys with corresponding disc addresses.
- To modify current storage devices to give a limited content addressing capability.

5.4.5. Data Flow Architectures.

The basic concept of computing is to perform one task at a time in a process. This is the 'Von Neumann Model' of sequential processing which must use intermediate memory locations.

Taking a computational process as an example:-

$$a * b + c * d + e * f$$

this would be processed by normal computer architecture in the following way:-

$$t_1 = a * b$$

$$t_2 = c * d$$

$$t_3 = e * f$$

$$t_4 = t_1 + t_2$$

$$\text{Result} = t_3 + t_4$$

In data flow, the computation would be performed as follows:-

Step 1 $a * b;\quad c * d;\quad e * f;$

Step 2 $a * b + c * d;\quad e * f;$

Step 3 $a * b + c * d + e * f.$

At each step the operations are executed in parallel. The time taken to perform the computation is the time taken to perform the 'critical path' operations whereas in existing architecture the time taken to perform the whole computation is the sum of the time taken to execute the constituents computations. This type of processing can be extended to the execution of any type of software including D.B.M.S. functions. If a particular privacy key required checking against a set of privacy locks the checking can be performed by several micro-processors in parallel.

6. APPROACHES TO DATA MANAGEMENT.

6.1. THE RELATIONAL DATA MODEL.

In early 1970 a man called Ed Codd published an article in the American computer magazine 'Communications of the Association of Computing Machinery' (C.A.C.M.) detailing a different approach to information retrieval called the Relational Model of Data. Relations, binary relations and the normalisation of relations have already been described in chapters 2 and 3.

The reasons given for the need for such a structure are:-

(1) This data model permits natural structures of data to be defined without super-imposed structures for machine representation.

(2) It forms the basis for data independence by overcoming 3 types of data dependency:-

(a) Ordering Dependencies.
The stored ordering is independent of order of presentation.

(b) Access Path Dependencies.
In both tree and network structuring, program and terminal activities are dependent on the available access path.

(c) Indexing Dependencies.
Indices vary depending on the activity being performed, programs then become dependent on the index alteration.

The relational data model removes these 3 dependencies by viewing the data as a table of values which is an entity/attribute matrix. This structure is natural without imposing additional structures for logical or physical representation.

6.1.1. Relational Model Accessing.

Using the following relations as example:-

EMPLOYEE (#-NAME, SALARY, MANAGER, DEPT.)
SALES (DEPT.,#-ITEM, VALUE)
SUPPLY (COMPANY, DEPT.,#- ITEM, VOLUME)
LOCATION (#-DEPT, FLOOR)
CLASS (#-ITEM, TYPE)

As an example of an information mapping using these relations, is as follows:-

$$\text{NAME}^{\text{EMP}}{}_{\text{DEPT}}\,('\text{TOY}')$$

This mapping consists of a relation name (EMP), a domain name (DEPT), a range name (NAME) and an argument ('TOY').

Mapping emulates the way people use tables i.e. in the above example the information required is to find names of employees in the 'TOY' department. The processing is quite extensive requiring looking down the DEPARTMENT column of the EMPLOYEE (EMP) relation to find entries containing the value 'TOY' and make a list of corresponding NAME entries.

A further example of an information request:-

'Find the names of employees who earn more than their managers'.

$$^{x}\,\text{NAME}\;\;\varepsilon\,\text{EMP}:\,x_{\text{SAL}} > {}^{\text{EMP}}_{\text{SAL}}\;\;\text{NAME}\;\;(^{x}\,\text{MANAGER})$$

This reads, 'there exists (ε) a set of names within the EMPLOYEE relation, such that (:), this salary (x) is greater than the salary of the employee who is a manager.

This example is of a 'greater than' test, there are several sets of operators in tests:-

numeric comparisons	: = ≠ < ≤ > ≥
set comparisons	: = ⊃ ⊇ ⊂ ⊆
arithmetic operators	: + - x /
set operators	: ∩ ∪ -
logical connectives	: ∧ ∨
parentheses for groupings	: ( )
built in functions	: SUM, COUNT, AVG, MAX, MIN, etc.

It is possible that when having found a set of values to not only examine them but to insert or delete such sets of values or even whole relations.

The basic characteristics of such a relational language are:-

(1) To be able to fetch a value or a set of values.

(2) To be able to change a value or set of values.

(3) To insert a data element or set of data elements into a relation.

(4) To delete a data element or set of data elements from a relation.

(5) To declare a relation and domains for inclusion in the established set of data base relations.

(6) To drop a relation from the data base.

In addition the language should give the capability of de-limiting the range of interaction between the user or his program and the data base.

A variable binding capability is also required to permit use in either interactive or a batch environment.

A name qualification is required to enable description of hierarchical structures i.e. relation name . attribute name.

Apart from a relational language acting in a self-contained manner, it should be possible to include such relational language statements as a sub-language within standard application languages such as PL/1 and COBOL.

The user should be able to obtain data in his working area in one of two ways:-

(1) In NON-PIPED mode, whereby all data is transferred to/from the work area in what appears to be several transfers taking place all at once.

(2) In PIPED mode whereby the user is given a tuple at a time.

6.2. THE CODASYL APPROACH.

The Conference on Data System Languages is a body which over the years has attempted to set a pattern or standard in certain areas of data processing. In the last 10 years major efforts have been made by this body in the development of a standard approach to data base management. This section describes briefly this approach to achieve a standard.

6.2.1. The Aims of Codasyl.

The stated aims of CODASYL were as follows:-

(1) To allow data to be structured in the most suitable way for application use regardless of the fact that some of that data may be used by other applications - flexibility is attained without redundancy.

(2) To permit the concurrent update and retrieval of data from a data base.

(3) To provide and permit the use of a variety of search strategies against the entire or part of the data base.

(4) To protect the data base against unauthorised access of data and from untoward interaction of programs.

(5) To provide a centralised capability to control physical placement of data.

(6) To provide device independence for programs.

(7) To allow for the declaration of a variety of data structures ranging from no connection between items of data to complex related structures.

(8) To permit the user to interact with data while being relieved of the mechanics of maintaining the structures.

(9) To allow programs to be as independent of data as current techniques permit.

(10) To provide for separate descriptions of data in the data base and of data known to the program.

(11) To provide a description of the data base not restricted to any particular processing language.

(12) To provide architecture permitting the description of the data base to enable the interfacing by multiple processing languages.

6.2.2. The Major Concepts of the CODASYL Approach.

The units of data structuring are as follows:-

Data Item.

The smallest unit of named data, an occurrence of which is a representation of a value.

Data Aggregate.

A named collection of data items within a record. There are 2 types of data aggregate:-

(1) A Vector - a one dimension, ordered collection of data items with the same characteristics.

(2) A Repeating Group - a collection of data that occurs an arbitrary number of times within a record occurrence. The grouping may consist of data items, and repeating groups.

Record.

A named collection (0→n) of data items and/or data aggregates. There may be an arbitrary number of occurrences in the data base for each record type specified.

Set.

A named collection of record types which establishes the characteristics of an arbitrary number of occurrences of the named set. Each set type must have one record type declared as its OWNER and one or more record types declared as its MEMBER records. Each set occurrence must contain 1 occurrence of its owner record and may contain an arbitrary number of occurrences of each of its member record types.

The following diagram (figure 6.1.) depicts a schema based on the set philosophy.

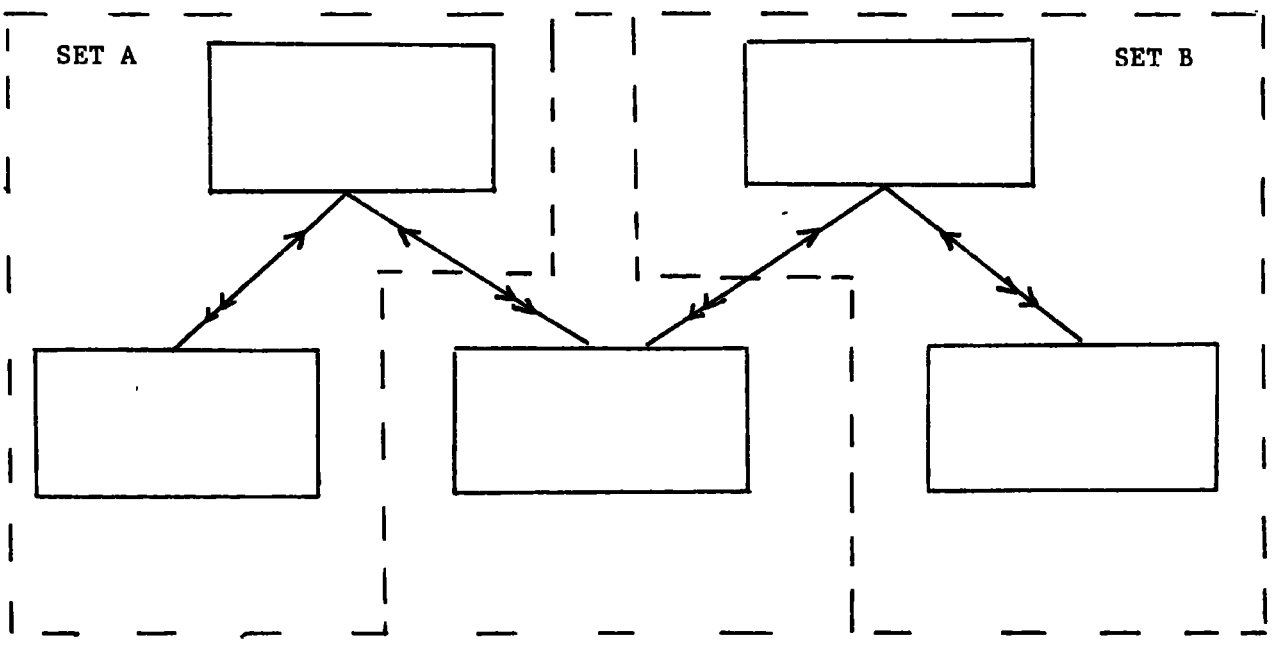


FIGURE 6.1. SCHEMA USING SETS.

Area.

An area is described as a named sub-division of addressable storage space in the data base and may contain occurrences of records or sets or parts of sets of various types. A TEMPORARY AREA is a storage area used by a particular transaction (run-unit) and destroyed at the end of transaction. It is possible to have as many temporary areas as there are current transactions. Areas can be opened by run-units for particular usage modes.

The area gives the D.B.A. the facility to sub-divide and treat parts of the data base as a single unit. The area permits placement control for efficient storage and retrieval.

Data Base.

All record occurrences, set occurrences and areas are controlled by a specific schema with the content of different data bases assumed disjoint.

Schema.

The schema is described by data description language entries including descriptions of all areas, set occurrences, record occurrences, data items and data aggregates.

Sub-schema.

The data description language entries which describe only those areas, sets, records, data items and data aggregates known to one or more programs. The descriptions are in the form known to those programs.

Data Management Language.

The statements used by the programmer to access the data base. The statements are a sub-language to the normal processing language such as COBOL.

6.2.3. Placement Control.

Placement control is based on the use of unique identifiers called DATA BASE KEYS. Each record is assigned a data base key. The assignment is at the first point of storage by the D.B.M.S. in accordance with:-

(1) the declaration for that record in the schema;

(2) arguments, if required, supplied by the run-unit which is adding the record to the data base.

The key remains until the record is deleted.

The data base key may be saved and subsequently used by the run-unit:-
(1) for direct access to the record;
(2) for reference later in the execution of the run-unit;
(3) for re-input to a subsequent run-unit in which the record is referenced.

It is possible to have a correlation between the data base key and the record's physical mapping.

There are facilities for optimising access time. The implementor can describe through the schema data description language (DDL) in which area a record is to be stored and also can inform the D.B.M.S. to store the record near another record or group of records to permit clustering of records liable to be used at the same time.

There are several placement/access techniques available:-
- placement WITHIN a specific AREA;
- by position within a SET i.e. the MEMBERS of the SET can be ordered such as FIRST, LAST, NEXT, PRIOR, immaterial and sorted;
- the calculation of the location by using the CALC qualifier with a programmer supplied procedure/algorithm to calculate the location.

To enable retrieval, the FIND command permits many different methods of finding records:-
- the run-unit supplies the data base key of the required record;
- the data base key is supplied using values supplied by the run-unit;
- the data base key is calculated using values extracted from the current record;
- by giving the position of the required record in relation to the current record;
- obtaining the current record of the given record type, set type or area;
- obtaining the owner of the current set of the given set type;
- by obtaining the first occurrence of the record matching the specifiers from the selected occurrence of a named set type.

There are no DML set manipulation commands, all such commands are based on record manipulation. It is possible to provide an access path down a hierarchy of sets by one of the following methods:-

- provide a top level set, the owner of which is the SYSTEM, there is only one occurrence of such an owner;

- the current set of the type sought is selected;

- the data base key of the owner record is provided by the run-unit;

- the data base key of the owner record is calculated using values supplied by the run-unit;

- by data base procedure;

The above options for placement, retrieval and access path control give a complicated variety of choice for the application programmer. A large number of variables and currency indicators must be initialised before execution of a DML verb. The use of such access methods is likely to be strictly controlled by the D.B.A.

There are facilities within the schema D.D.L. to logically include and protect from destruction a record occurrence within a set. This is achieved by giving a particular record type a membership class within a set type:-

AUTOMATIC means that membership of the set is established by the D.B.M.S. as soon as the record occurrence is stored.

MANUAL has the opposite meaning, that an application programmer must use a DML verb to insert a record in the set.

MANDATORY means that a member record once a member of a set cannot be removed until its owner is removed.

OPTIONAL means that a record can be removed from a set without removing the set's owner.

6.2.4. Privacy and Integrity Controls.

Privacy locks are provided at several different data levels at the schema, sub-schema, area, record, set, aggregate and item levels. The programmer must supply the key when attempting to access any of these data levels where a lock exists.

The locks for schema and sub-schema are to restrict alteration or display of the schema/sub-schema, restrict display of locks and to restrict the use of the sub-schema for compilation.

The locks at the area level restrict retrieval or update from the area and restrict the use of any of the support functions on the area.

Record, data aggregate and data item locks restrict the use of each of the DML commands which apply to the record.

Set locks restrict DML commands operating on set arguments.

Privacy keys and locks are constants, variables or the result of a procedure.

Integrity controls are provided to prevent concurrent accessing of the same data. This problem has been described in chapter 3, CODASYL overcomes these concurrency problems by permitting EXCLUSIVE control (read or update) when opening an area. PROTECTED control is also provided to permit concurrent retrieval but not concurrent update.

6.2.5. Criticisms of CODASYL Approach.

Two major criticisms have been raised (particularly by I.B.M.) concerning the proposals:-

(1) The proposals possess poor data independence, the programmer needs to know several aspects of the data base structure, the programmer must know in which area a record is placed and must know the methods of access and placement so that the appropriate variables can be initialised. Once the programmer knows the data base keys the records can be accessed without using the D.B.M.S.

(2) The languages, particularly the DML are too complex. There can be undetected programmer errors with the setting or suppression of currency indicators.

Much of the detail of this section appears in reference 7. The proposals have formed the basis of approach for several D.B.M.S., notably D.M.S. 1100 and I.D.M.S. The proposals are an attempt at a standard to direct data base developments. Proposals which are constantly under review for improvement since the first proposals were published in 1971.

6.3. THE GUIDE/SHARE REQUIREMENTS.

GUIDE and SHARE are two I.B.M. user groups. The two groups formed a committee to produce a definition of long range requirements for data base management systems. To quote from the report "The report does not describe an existing system, but does present a viable operating environment in which a data base management system can be expected to operate. The significance of the report lies in the fact that it represents the data base requirements of a large class of influential and knowledgeable users." (Reference 8.)

The basic set of requirements are:-

(1) Data Independence.

(2) Data Relatability.

(3) Data non-reduncancy.

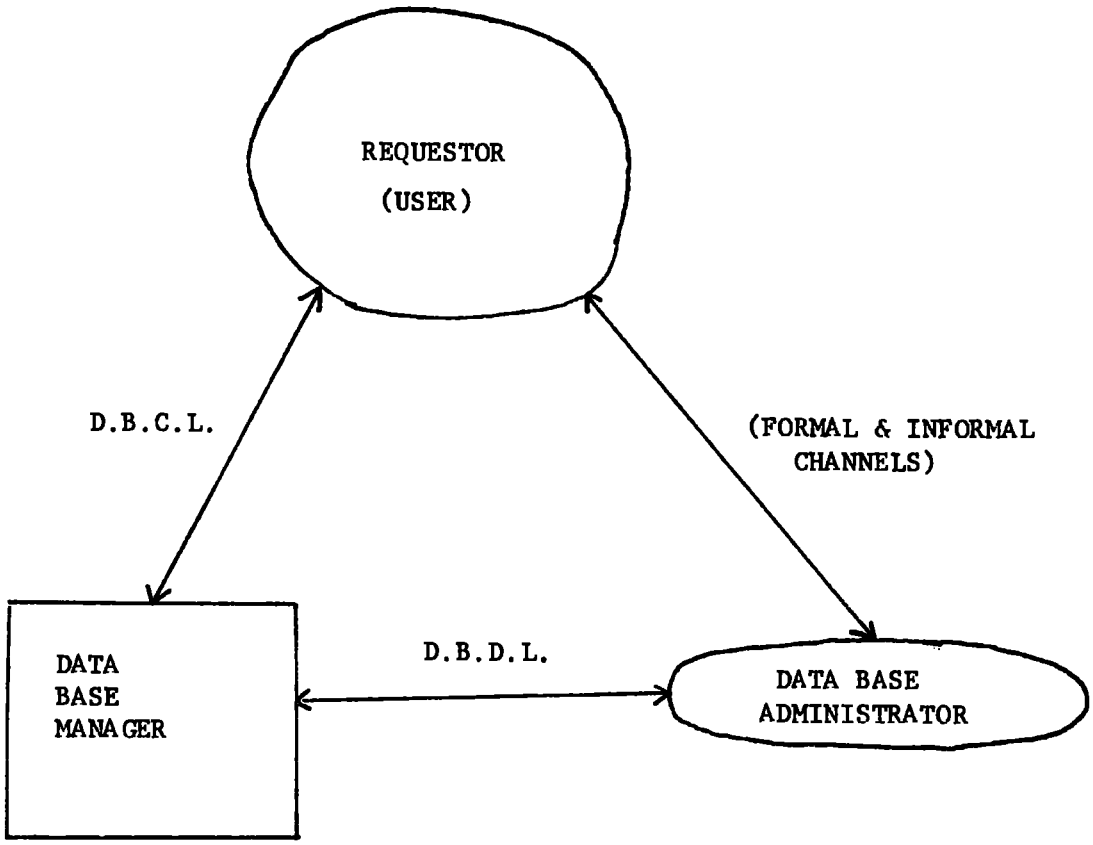
(4) Data Integrity.

(5) Security.

(6) Performance.

(7) Compatibility with existing concepts.

The requirements split the data base environment into 3 principal functions as set out in Figure 6.2.



D.B.C.L. - Data Base Command Language.

D.B.D.L. - Data Base Descriptive Language.

FIGURE 6.2. THREE PRINCIPAL FUNCTIONS OF THE D.B.M.S.

The requestor (a title to include both applications programmer and information specialist) is a person who makes available information to the enterprise based on data stored in the data base. To do this, the data must have the necessary level of independence, relatability and non-redundancy, as well as having extensive end-user facilities.

6.3.1. The Data Base Administrator.

The requirements lay down a set of functions that the administrator must perform:-

(1) The description of data.
(2) The definition of relationships.
(3) The definition of mappings.
(4) The establishment of data security rules.
(5) The specification of performance measurement procedures.

These points have already been covered in Chapter 5.

The levels of data structure that GUIDE/SHARE suggest are:-

Entity Types.

The entity record is the 'unit of information' (the schema record in Chapter 2.)

Physical Data Structures.

These structures consist of what are called 'units of system data' or 'data records' equivalent to what have already been defined as the physically stored records.

Logical Data Structures.

These structures are 'units of application data' or 'logical records' equivalent to sub-schema records.

The need to set up a 'logical mapping' between logical record and entity record and a 'physical mapping' between entity record and data record was defined as a requirement to bridge the 3-level structure to form a data accessing structure.

6.3.2. The Data Base Manager.

The Data Base Manager (D.B.M.) is a "collection of hardware and software resources that serve as the interface between the data base and its users" (Reference 8).

The primary function is to access data as a result of a user request and to transform the data from the physically stored format to the logical format required by the user. Figure 6.3. describes the position of the Data Base Manager.

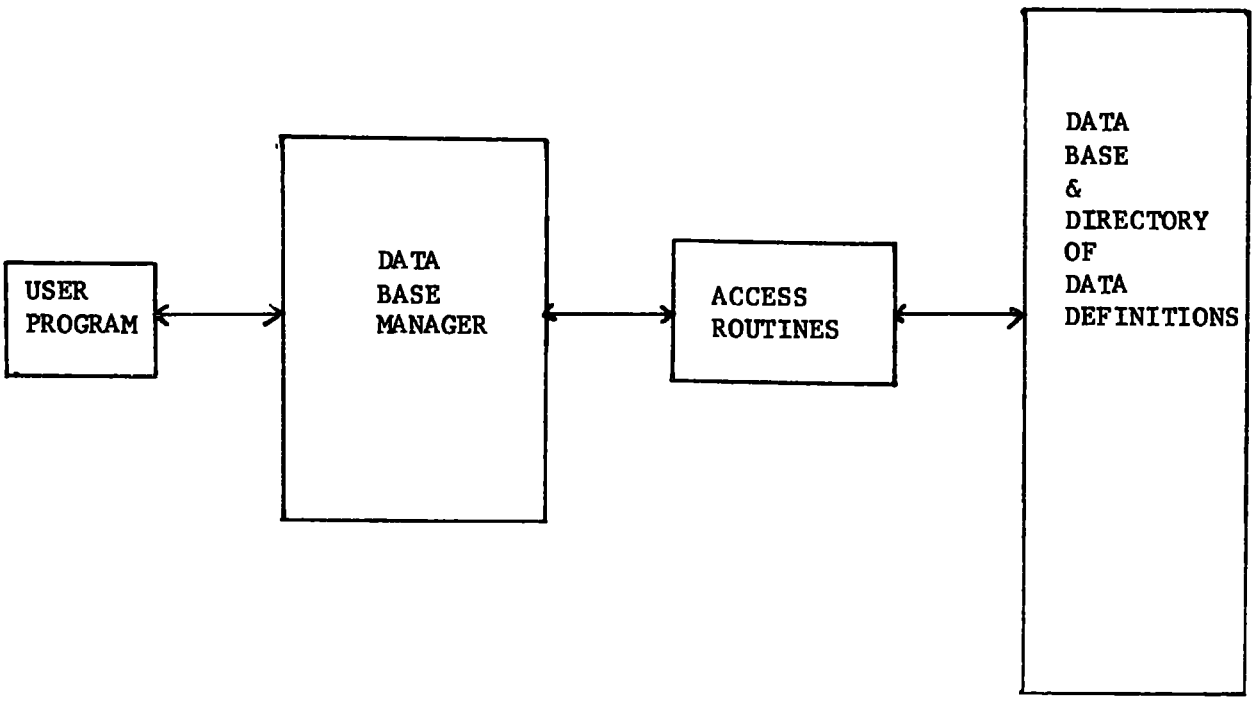


FIGURE 6.3. THE DATA MANIPULATION FUNCTION.

The D.B.M. must ensure data security is maintained, must provide restart facilities and maintain and record performance levels.

6.3.3. The Data Base Descriptive Language.

A non-procedural and extensible language that permits the D.B.A. to specify his data requirements to the data base management system and in particular the data base manager component.

The functional capabilities should include at least the following:-

(1) A means of describing the logical and physical properties of data structures.

(2) A means of describing relationships.

(3) A means of describing mapping rules.
(4) A means of specifying security procedures and access rules.
(5) A means of subsetting the data base to minimise access times and control costs.
(6) A means of specifying monitoring requirements.

As far as logical data relationships are concerned there is the necessity for certain types of logical data to be related:-
(1) Between two data items at the same level.
(2) Between two data units at different hierarchy levels.
(3) Between a data item and an entity field.
(4) Between a data item and its corresponding value.

Other requirements with regard to logical data are:-
(1) The facility to define conditional and unconditional relationships.
(2) The facility to define hierarchical relationships including the definition of repeating groups and elementary data items.
(3) The facility for specifying membership rules.
(4) The facility for denoting operational rules for file operations. (i.e. additions and deletions.)
(5) The facility for specifying sequencing rules.
(6) The facility for defining logical relationships between files.
(7) The facility for establishing logical data relationships from entity descriptors.
(8) The facility to support, supplement and not conflict with host language characteristics and relevant data structures.

At the entity level, the following are requirements:-
(1) The facility to distinguish between real and virtual fields.
(2) The facility of defining an entity to be simultaneously a component of two or more entity groups, constructs, record types or any combination thereof.
(3) The facility for specifying ordering attributes.
(4) The facility of defining mappings and relationships between units of entity data.
(5) The facility of associating operational functions with physical data structures.
(6) The facility for describing hierarchical structures.

At the third level the physical data description should be able to be described with the following characteristics:-

(1) The capability for having initial, default and null values.

(2) Compaction algorithms should be provided.

(3) Ordering attributes for elements in records, records in data sets or extents.

(4) Indices.

(5) Conditions under which a specific record becomes a member of an extent or data set.

(6) Physical data set attributes, such as block size, access method, space allocation and device allocation.

(7) Retrieval strategies and mapping facilities.

(8) Identifying redundant data.

(9) Derived data elements.

6.3.4. Data Base Command Language.

A very simple command language is suggested as follows:-

(1) Open logical file - OPEN.

(2) Retrieve logical records - RETRIEVE.

(3) Add logical records to the data base - INSERT.

(4) Return logical records to the data base - RETURN.

(5) Delete logical records - DELETE.

(6) Close logical file - CLOSE.

The commands have associated with them a parameter list providing operational data, qualifiers and default conditions. Logical records may be accessed by identifier or by a sequence indicator. Typical sequence qualifiers are:- next, previous, unique, first and last. It should be possible to have Boolean and relational operations as part of the parameter list. Defaults may be supplied by the system or by the user.

6.3.5. Miscellaneous GUIDE/SHARE D.B.M.S. Requirements.

There are several characteristics that the group list which are not directly related to the accessing or description of data.

Data Transformation.

This is the capability to alter the form of a data item for use by an application program from the form of the data item described in the data base directory.

They state that the D.B.M. must be able to derive virtual data.

Modularity.

The data base manager should be modular. The D.B.A. should be able to select a usable, proper and compatible subset of D.B.M. functions to be able to configure the D.B.M. to a particular installation's needs.

Extensibility.

The D.B.M.S. must be designed in such a way as to provide for evolutionary changes without requiring any major conversions.

Re-configuration.

The D.B.M. must be able to respond to hardware reconfiguration defined and invoked by the D.B.A.

D.B.M. Utilities.

The D.B.M. requires the support of specialised system utilities designed to assist the D.B.A. and the installation in the maintenance of the data base as defined by the D.D.L.

Examples of the facilities required:-

- Building and updating the D.D.L.
- Reorganising the data base when physical structures are changed.
- Displaying the contents of the D.D.L.
- Copying various physical units of data from one medium to another.
- Displaying various physical units of data.
- Printing all or selected parts of the D.B.M. system log.

Such utilities should be able to run concurrently with application programs accessing the data base.

D.B.M. Networks.

A D.B.M. must allow different D.B.M's. to communicate with it whether operating on different machines at the same or remote locations in a manner transparent to the application program D.B.M.

Own Code.

The D.B.A. should be able to add 'own code' modules to the D.B.M.S.

Error Detection, Correction and Recovery.

The D.B.A. should be able to provide installation-defined routines to edit and validate occurrences of units of data for proof of their integrity.

The D.B.A. should be able to request the automatic generation of a system log.

In order to correct discovered abnormalities, the D.B.A. must have suitable data recovery techniques that are under control of the D.B.M.

The D.B.M. must provide the D.B.A. with the facility to lock out portions of the data base until those portions have been repaired or restored.

The D.B.A. must have the ability to bypass D.B.M. controls to allow for the display of, or modification to linkages which may have been damaged.

Response to Abnormal Conditions.

The D.B.M. itself should never abnormally terminate as a result of an A.P. error.

Depending on the severity of the problem, the D.B.M. may according to rules defined by the D.B.A:-

- Return indicators to the user program concerning the status of the operation performed or attempted.
- Terminate a user program.
- Cut off access to a part of the data base.

Status indicators are returned to the applications program notifying the user that some abnormal situation has occurred, some examples are:-

- space extents exceeded;
- data item reference does not exist;
- user has no authority to access the data element specified;
- inconvertible format specified in a user program;
- null data element accessed;

- logical record does not exist on file;
- an attempt made to insert a logical record into a file when a logical record with the same identifier already exists;
- an attempt to delete a non-existent record from a file.

User Test Facilities.

The D.B.M. must have a 'test mode' operation so that the real data is protected against accidental destruction when manipulated during the testing phase.

The D.B.M. should provide a number of other testing features, some examples are:-

- debugging and dumping aids;
- access trace aids;
- a selective file copy routine;
- aids for checking equality or parity of ' parallel operations';
- a test file data generator.

D.B.A. Control Facilities.

Under the direction of the D.B.A., the D.B.M. will be able to select the most appropriate medium for the residence of physical data.

The D.B.A. should have the capability to instruct the D.B.M. to select or change the physical storage structure of data.

Test files should be able to be set up by utilities.

Global flags should be available to the D.B.A. to restrict usage of data to enable reconstruction, reorganisation.

The D.B.A. must be able to display data base descriptors and their attributes.

Conflicting Requests for Data.

The D.B.M. must be able to protect the system against deadlocks in a manner that allows reasonable freedom to users without jeopardising the integrity of the data base.

Compatibility.

An installation's program's written to execute in a pre-data base environment must be able to execute within the framework of the D.B.M.S.

User's Guide.

A comprehensive user's guide should be provided to assist users in making decisions concerning their particular use of the system.

The user's guide should contain definite guidelines for a number of different hardware configurations.

6.4. REFERENCES.

7. The IBM Position on the CODASYL DBTG Proposal.
   Dr. T.R. Heywood, IBM UK Laboratories.

8. The GUIDE/SHARE Data Base Requirements.
   P.H. Prowse, Chairman
   GUIDE/SHARE Working Group.

(Both papers from the BCS October 1971 Conference on Data Bases.)

PART TWO

7. DATA MANAGEMENT NEED THROUGH IMPLEMENTATION.

7.1. TODAY'S PHILOSOPHIES ARE INADEQUATE FOR TOMORROW.

Present day data processing (Management Services) departments on the whole provide an inadequate service to their enterprises' user departments. This is true of the majority of installations to varying degrees within the different user sectors within Britain. The standing of computers in the outside world is poor, regularly the butt of an enterprise's poor administration or management. Not many years ago the data processing service was adequate but this service now lags behind the technology and the user requirement. There are 3 main reasons for this fall in standards.

(1) A lack of education in every aspect of information technology. Management at all levels of the data processing service have been too inward looking. The need is for users to be educated. This education and enthusiasm is required from the top down in user management, provided by all levels of D.P. management and staff whenever in contact with users. Informed users will generate their own enthusiasm over the use of this new technology and set the pace in the development of their own requirements, then the problem for the D.P. department is to control the flood of user requirements. Fear of technology springs from a lack of education and understanding. The requirement for D.P. staff and management to maintain their own education level as new technologies appear is equally important. D.P. management must motivate their staff to achieve this exercise.

(2) The organisation of the D.P. department is not the straight forward exercise it was several years ago. It is vital that the organisation should evolve as requirements change. Management must in such circumstances be flexible. With the advances in technology and the extra possibilities available to users, extra depth to D.P. management skills is required, with the awareness of technical possibilities coupled with the user situation, to provide a cost effective solution to any problem. Current applications are becoming more complex to implement. The question to be asked is have the task and personnel management skills developed to the necessary level? In the majority of installations this cannot be true. As more installations develop, the demand for good D.P. management outstrips supply. From this realisation springs the facilities management and turnkey systems market with outside service organisations providing users with a complete off-the-shelf installation.

The D.P. services market is growing faster than the sale of hardware. Professionalism is the important word, the importance of formal structuring of programs to develop the most effective methods of system design and implementation, enabling systems to be costed more accurately, to be produced on time, and to be easily changed. A more professional approach, normal practice in other types of engineering. Because the discipline is in its early days, new technology advances overtake themselves, a major reason for the short life of computer systems. In the future a much greater effort is required by users to protect development investment and produce systems that last. Computer manufacturers see this problem as an attack on their own customer base and bridging hardware/software is provided by them to protect the user's system's investment and so protect their own customer base. Previous chapters showed that the D.B.M.S., in attempts to provide systems immunity from structural and value changes in data, gives assistance to this task but not the total solution, much greater thought must be given by users to their system design to enable long-life. Testing of systems is abysmal. The tools to test systems must improve in the future, but that is no excuse for the attitude in many D.P. installations, that of running a system in a live situation with only meagre testing done and hoping to correct the expectant fault when it occurs. Often the reason is that time is money but this situation cannot be allowed to continue. There is of course no need to go to the extremes in the aircraft industry, where life is at stake, but computer systems testing falls short 'by a mile'.

(3) The D.P. management's handling of new technology must improve in the future. The golden rule correctly practised in most installations is, do not be first in a new software/hardware package or new technology. Someone must be first and there may be good reasons to be first, but in such circumstances management must realise the heavy cost in manpower and money that will inevitably be incurred. The worst failing in the D.P. management's handling of new developments is that they become over ambitious in the use of new technology. The past is strewn with examples of over ambitious projects that have become 'white elephants'. Management should be, if anything, conservative in the use of new technology but with a projection into the future to define what may be possible in the years ahead.

The problems that face management now must be faced and gradually overcome, because the use of D.B.M.S. is a tool which will completely alter the design and operation of systems in the future. It will cause a revolution in data processing. As the comment in Chapter 1 suggests, the phrase data processing is outmoded because the processing of data is no longer central, it is the data as an information source that is central to what one might call an Information Service, or the popular term Management Service, rather than Data Processing Service.

7.2. FUNCTIONAL OR STRATEGIC?

When an organisation first considers the possibility of using a D.B.M.S. the first decision the top data processing executive (the Management Service Manager) must make is whether to consider the use of the D.B.M.S. in a functional or a strategic manner. The difference in approach will make a vital difference to the long term success of the data base venture.

The functional approach is based on the idea that the data base package has been selected for a particular application or application area. Primarily the reason for the functional selection of a D.B.M.S. is to enable complex data accessing e.g. chaining of data for a particular application. There is nothing wrong with this approach, the purchase price of £20,000 for a D.B.M.S. may pay for itself in manpower cost saving in the development of such a complex structured application. There are many examples of in-house non-D.B.M.S. systems costing 5-figures that are cumbersome with tardily responsive terminal facilities. In such cases the use of a D.B.M.S. from the earliest design stages could have provided a streamlined and responsive system. Many D.B.M.S. systems currently on the market do provide good data access and performance, indeed its a commonly held view that current data base packages are nothing more than software providing good access handling. With the current state of the art, this is not necessarily an incorrect stance to take as far as current implementations are concerned, but this thesis is, and the Management Service Manager should be,concerned with the more strategic stance for long term planning.

If current data base software does not inspire the manager to take a strategic long term view, he must remember that current D.B.M.S. is first generation and is only a tool in the management of the organisation's data resource. What is

the meaning of the strategic approach? Simply, to regard the future use of D.B.M.S. as an environment rather than a development package in the same way that the current operating system software provides an environment. The way that future data base systems will develop, with the inclusion of specialised hardware as part of the D.B.M.S., will force management to think of the management of data strategically rather than functionally. This presents important ramifications as to how the data base products will change. There will be a polarisation of D.B.M.S. towards specific manufacturers, data base systems and operating systems will relate much more closely together. The operating systems for the ICL 2900 series are conceptually much more aware of the need to manage data, eventually, no doubt, packages such I.D.M.S. will be replaced or radically changed to be enveloped within the machine's operating environment with fully integrated data resource management including dedicated processing hardware to search the data bases. The use of proprietary D.B.M.S. software will in the future slowly fade away. The stance of I.B.M. in such future developments will be important. I.M.S., their current main data product, will disappear as soon as possible with the development of machine architecture and hardware/software geared to the management of data rather than currently obtaining the best throughput.

This will be the manufacturer's contribution to treating data management as a strategic exercise, but it is up to the Management Service management to think strategically in the long term. The management can think strategically even with today's D.B.M.S. software technology.

At the physical storage level, this means having subject data bases rather than application data bases. Data base files containing customer details, invoices, parts, not data bases functionally built up with purchase control or production control data. The most elementary initial data base application may require several such subject data bases and the temptation is to go for a single application related data base so that the initial project will be implemented quickly, easily and profitably for user management. The inevitable result after the initial euphoria is that when the time comes to build on the start made, the data will have to be physically restructured, data independence will have been drastically reduced, the physically stored data becomes a complex muddle to permit the extension to the existing application or the development of a related application, and allowances are continually made to accommodate changes producing an integrated morass with poor performance and response times. The cost of an

early short cut is the data base'quick sand'. It may seem inefficient to design the first simple system to access maybe 3 separate files when one will do, but the benefits will be derived later. The next application to use the data base approach may be found to require another two subject data base files, but gradually more and more applications can be developed using the same set of data base files, indeed all the application systems an organisation can think of to develop, may well only require about ten such files. This strategic method, where the data base files are independent of the applications that use them, has far reaching advantages. A great reduction in data redundancy is possible, the basis of data independence is formed and integration of applications can be better controlled and easier to implement. A steady growth of systems becomes possible on which can be founded a complete information strategy.

7.3. MANAGEMENT COMPREHENSION AND COMMITMENT.

There are many barriers that must be broken down when Management Service management approach their users to persuade the user of the benefits of the data base. The initial approach should be made by the Management Service Manager or System Development Manager at user management level. By convincing the top level management of the importance of the concept makes the conversion of lower level management and staff to the data base approach much easier, and without the agreement of top level management such 'winning over' of the rest of the organisation is impossible.

The first necessity for Management Services to realise, is the necessity to understand the fears of user management and sweep away those fears generating an enthusiasm for the benefits of the new approach. The initial exploratory talks are first and foremost a selling exercise.

An enterprise's organisation chart over-simplifies relationships between different managers and different departments. In order to survive, managers become posessive about their information, there are inter-departmental rivalries. The sharing, albeit controlled sharing of data will break down departmental barriers, the D.P. executive must break down the political barriers by persuasive talking.

There is the fear of user management and staff that they may be inadequate to change the mode of their working. If the manager or staff are old it may be very difficult to change their working ways.

Decision makers in organisations tend to be of two kinds:-

(1) The organisers.
(2) Creative men with ideas but little organisational ability.

The two types of decision maker do not work well together. The user manager will be primarily of the first type the data base designer the second type. Although the distinction is not clear cut, it is rare that a decision maker will have a correct balance between creativity and organising. The D.P. executive must attempt to bridge this gap between staff, the data base designer (ideas man) and the user manager (primarily an organiser). If the bridge is not made from the start, the user manager may through fear, build barriers to prevent the creative data base designer from doing an effective job. Any hint that the user manager or staff may loose power or freedom will result in failure. Status and income levels of all management and staff must be assured. Each manager has his own style that must be protected.

It will not be easy for the D.P. executive to obtain a clear indication of whether he has succeeded to 'win over' the user manager's support. Such user co-operation cannot be 100 per cent. If such a level co-operation is being offered the D.P. executive may be being misled. Lack of co-operation can be positive and clearly stated or, possibly worse and more likely, the lack of co-operation will be of a negative form. Problems may be created that are meant to disrupt but which when explained to the D.P. executive are expressed by the user in a way to suggest that no other option was available.

Top level executive management may be found by the D.P. executive to be fairly accepting of the data base concept with the assurance of flexible and timely management information. The departmental management with departmental rivalries may take a while to be won to the cause, but there are three important factors in gradually attaining support.

(1) All information that concerns a particular manager must be passed to that manager without the jargon-filled talk of the D.P. project leader. As developments are made and explained in clear English the communication barrier is broken down and understanding prevails.

(2) Participation from user management and staff is vital. For any project a committee should be formed of D.P. systems development and user staff, to enable users views to be expressed. To permit a controlled degree of design by the users themselves can only benefit relationships. Its the feeling that the user has some control over his destiny which breaks down mistrust. The participation must be seen to be real otherwise more resentment will be built up.

(3) Education must be at the centre of all dealing with all levels of user. All levels of staff must share in the benefits of the new system.

7.4. INFRASTRUCTURE.

The history of implementation of D.B.M.S. has been frought with disaster and has been accompanied by a singular lack of success. To put it succinctly, to quote from James Martin (Reference 9) "Many organisations which had the right software, brilliant implementors, and a big budget met with disaster because they were charging windmills". It may be because they ignored, or were not successful in the pursuit of user management acceptance and involvement as described in section 7.3. The implementors (D.P. management) may have been under several misconceptions.

A D.B.M.S. is not a management information system. The D.B.M.S. is a tool in the progressive development of operation systems based on a data-oriented environment which will permit the gradual evolution of a management information system.

The data base will not hold all the enterprise's data. To start with this basic objective will doom the implementation to disaster. To hold all the enterprise's data within the data base, implies either a great deal of data redundancy or a great deal of integration amongst application systems. The need is not to seek or force integration but to permit it naturally and then only if performance levels will not be drastically affected. The best implementations start slowly and build in effectiveness over the years.

Following on from the above point, an organisation will not have simply one data base but several. One important point to distinguish is, what is a data base? The scope of meaning that it embraces varies from installation to installation. The best definition is probably "all the data that is involved in the access of a single schema, it is the scope of data referenced by that

schema". It includes all the basic subject files as well as all the schema data used in the access of the subject files (e.g. index tables, access algorithms used in the access methods and security/recovery data files). The schema is the global map of data integration and the data base is the collective unit of data described by a single schema. It is possible to have several schema's varying in complexity (they should be simple at first) giving the minimum data integration needed to provide the optimum information needs of the organisation. In this sense, a single subject file (a customer file for instance) could be used in separate schemas and by definition separate data bases. It is better than an enterprise's schemas start simply, growing and interlinking over the years without the necessity of rewriting application programs and without the need to change when hardware changes. In this way the D.B.M.S. forms an infrastructure which permits this growth.

Central to such success is standardisation in certain areas:-

(1) The need for a single data description language by which all of an enterprise's data is described irrespective of the machine on which the data may be held or the geographical location of the data. Similarly there must be a single data dictionary system employed to document the enterprise's data.

(2) A single corporate-wide (and therefore centrally controlled) organising principle which forms the structure for data base development. Central to the achievement of this organising principle is the eventual appointment of a Data Base Administrator.

The growth in the development of the data base environment must be on an incremental basis with one application and one improvement implemented at one time. Even the best laid plans go astray but this is only to be expected.

7.4.1. <u>The Initial Data Base Projects</u>.

The first application should be chosen on the basis of three characteristics:-

(1) The data base should not be too complicated.

(2) The application should be one that is cost justifiable.

(3) The operating managements of the user departments should give full support and co-operation.

The first point emphasises the need to have a project that is simple for the inexperienced data base development staff to enable the gaining of experience. This project is critical with emphasis on implementing the project as quickly as possible (it must be within a year of conception) and the training of staff.

The final two points emphasise the need to benefit the user. Cost justification on a data base project is not easy because most of the benefits are intangible. The true benefits of the data base environment can be seen by viewing all the data base applications as a whole over a long time scale rather than comparing the merits of using the data base approach for a single project. Certainly it is very important that the user really wants to use the first application and the user's involvement is very necessary.

A data base project does not necessarily imply the use of on-line access, there are countless worthwhile batch data base projects, but certainly the benefits of the user having immediate access seems to suggest greater benefits even though such benefits have nothing really to do with using a D.B.M.S. A succession of short, sharp projects (all less than 1 year to implement) should follow the initial 'success'.

7.4.2. Long Term Planning.

It is important that while the initial projects are being developed a great deal of thought is given to long term planning. It is pointless to have developed many very neat but small projects then to realise when this phase is complete that no thought has been given to the future large long term projects. From the very beginning the organisation must know the big projects that eventually it will wish to undertake. From the very start there must be an undertaking to move towards the implementation of these large integrated systems, and the initial short projects described above will be a useful step in that direction. The small projects must be pointing in the right direction enabling them to be built upon gradually but showing the users that developments are constantly continuing. It would be totally frustrating for users if, with the small projects out of the way, developments start on the large new integrated systems which are in totally different application areas to those small projects already developed. Worse still should the large scale developments mean that the initial projects are replaced. A failure of this nature will cause a large

time gap (and no doubt a large credibility gap) between the implementation of the first phase and the implementation of subsequent phases. No doubt during this second phase the initial applications may be altered as they are integrated into the larger systems. The end of this first phase is clearly a 'watershed'. The opportunity should be taken to fully review the progress, the success and failures during this first crucial step into data management. The second phase will contain the major and more complex design for the large project. Nevertheless the implementation of the large project as a whole should consist of a step-by-step implementation continuing the incremental growth pattern, using the first phase as the basis for this growth.

While the initial phase applications are being developed, part of the project team which is responsible for implementation of the whole project should be performing the long term data analysis exercise for the whole project, to define the 'real world' data and how it is used within the departments concerned. The objectives of the development staff in this phase should be as follows:-

(1) To define the departments that are involved in the full integrated system.
(2) To meet the departmental management and staff at all levels.
(3) To understand and document the methods of working at levels.
(4) To define the flow of information both within the department and across departmental boundaries. Departments are often sensitive about their own information being passed to other departments.
(5) To 'separate the wood from the trees.' To define the data entities and attributes of the entities that the operation of the departments are based upon. The relationships between the entities and attributes must be meticulously documented. The properties and characteristics of these relationships should be also documented giving particular attention to the degree of the relationships (one-to-many, many-to-many etc.).
(6) The drawing up of a chart describing these real world entities and their relationships with possibly the placement of the chart on the project office wall. A specification should be produced documenting the entities, attributes, relationships, the data flow and the relative volume of data flow. There may be the necessity to have a chart showing the flow of data within and across departments.

7.4.3. System Development Staffing for the First Phase.

There are two theories of management described by Douglas M. McGregor as 'Theory X' and 'Theory Y'. The former is a highly mechanical approach to management, the latter a very flexible and organic approach. The best approach is a mixture of the two, however in this initial phase, where a a new concept is involved for the first time, education is important and a flexible approach is necessary, with the project team members having varied roles. The need for project control and leadership is still paramount but it is probably best if the roles of the team members are not too distinct. There must be more mechanisation and more clearly defined roles in the subsequent phases.

Each large project should have a project team. One other project leader should be directly responsible for technical aspects i.e. implementation of the D.B.M.S. and data communication facilities. The members of this team may well be required to provide such things as data base development aids, but during this first phase, the emphasis must be on a quick implementation and members of this team should be prepared to work within the other project teams to achieve the implementation objectives. These technical project team members should be prepared to be working on the most time critical as well as technically critical aspects of the implementation. The D.B.M.S. should be used for the small stand-alone projects as well as the larger integrated developments. Once implemented these operation systems both large and small will form the basis for the development of any management information systems that may be required in the future.

In the first phase the project leaders of the individual large projects will be responsible for both the implementation of the initial short term implementations and also he/she must oversee the departmental data analysis described above. This latter part of the project will probably be undertaken by the senior analyst within the team. If the user management had been convinced of the benefits of the data base approach during the initial meetings, the senior analyst should find his task easy with the user staff providing enthusiastic suggestions, some useful others irrelevant. There may be many suggestions from user management for management information facilities, the analyst should nod understandingly making notes of what they consider are their requirements without dampening their enthusiasm by explaining that such facilities are some years away.

It is important that there are regular meetings between members of the different project teams to enable cross-fertilisation of ideas so that the education process can progress as rapidly as possible during this initial phase.

7.5. THE MACHINE ENVIRONMENT.

7.5.1. Present Hardware.

Under the machine covers the hardware technology has changed swiftly over the last few years. New degrees of minuturisation with Large Scale Integrated Circuitry enabling more computer power in a smaller sized machine. Semi-conductor storage is now common and with new developments in electronic storage occurring regularly, the capacity of electronic storage possible on a single computer system is increasing. New architectural techniques are being implemented with the virtual machine environment gaining prominence. Micro-code is introducing new flexibility at the hardware level. The capacity of electro-mechanical storage devices are also increasing with a single magnetic disc cartridge capable of holding several hundred million characters. Facilities may have been extended but the fundamentals of the way mainframes are being used does not differ to any great extent from 5 or 10 years ago. Probably one of the biggest of impacts on users is the increased power and facilities of mini-computers for much the same price, making them a cost effective alternative to the expensive-in-comparison mainframe. Possibly even more startling has been the extension to user terminal facilities. The availability of programmed intelligence and some degree of terminal storage such as casette or floppy disc, permitting a user to use his terminal in isolation (off-line) from the mainframe for certain tasks and so saving line costs.

7.5.2. Methods of Operation.

There is a lot of talk about the impact that the minicomputer and micro-computer revolutions are going to have on the mainframe market. In computer bureaux and universities the need for the mainframe is ever present. The need for remote batch and multi-access facilities requires flexibility of operation which inevitably means a large operating system on a large mainframe machine.

Most organisations, large or small do not require such a degree of flexibility. The requirements of such an organisation are:-

(1) Facilities for system development, compilers, testing aids, test date generators etc.
(2) A facility to run batch production work.
(3) A facility, for up to a known maximum of on-line access, to retrieve and update data stored on the computer system.
(4) The possibility, in a production or laboratory environment, for the need for real time computer access to control machinery or laboratory equipment.

A large part of the next chapter considers the need for a D.B.M.S. to be able to handle, access and control data in each one of these 4 modes of operation.

System Development.

A D.B.M.S. must cater for the needs of the programmer and analyst, it must provide a testing environment.

Batch.

The easiest of all modes for a D.B.M.S. to handle.

On-line.

The fastest growing mode of operation. There are two types of on-line access:-

(1) Transaction Processing. Transaction processing is a type of on-line access that usually has a specific response time requirement. It is not necessary that a certain response time be met every time but certainly it is expected that the vast majority of responses will be within a specific response time band which will probably be less than 10 seconds. This is different to the need to ALWAYS meet response times in proper real time systems where equipment is being controlled. Most operation on-line systems are of this nature. The major problem which causes fluctuations in response times is undoubtedly poor file accessing. The problems are magnified when data is being updated on-line with the need for security against concurrent updating. The D.B.M.S. faces its stiffest task when used in this operating mode.

(2) On-line access not requiring a specific response time. It is usual that such systems are enquiry systems with the terminal in use only periodically. The response time is still important, there should still be a response time of only seconds, but because the operator is not at the terminal all day, fluctuations in response do not have such an annoying affect. Terminal access to a management information system would operate in this mode. The actual search for information may take minutes or hours and may be performed in batch mode although the search may have commenced from a terminal. Information concerning pre-search statistics prior to the management information request should nevertheless be returned in a reasonable time interval.

Real Time.

Because a response time must invariably be within a certain interval, this differentiates this mode from the standard commercial transaction processing system described above. The response must be always on time. The proper real time control system can never really have too fast a response time. It is the type of operation where performance is important above all else. The use of D.B.M.S. in such an environment to date is fairly restricted because the D.B.M.S. with its generality is not suited to such a mode.

In conclusion on the subject of operating modes, the operating system of the past with its vast flexibility and complexity is no longer needed. The type of operating system that absorbs 70% or 80% of the machine's execution time should be a thing of the past. The mainframe operating system of the future should be data rather than process oriented, providing streamlined data base management and terminal control. The future operating system should be nothing more than a super executive interfacing the D.B.M.S. and the data communication's network of the future to the mainframe's sophisticated architecture.

7.5.3. Future Hardware Developments.

The Management Services Manager must be aware of the possible development in technology that might affect his future plans. He must be aware of the variety of future developments which may reach the market place so that he can keep his options open as his own enterprise's data processing activities develop.

In the previous section a suggestion was made that new developments in architecture will be welcomed by the user but that less importance will be attached to sophisticated operating system facilities. At present one of the main differences between mini and mainframe is the level of operating system software facilities. As this significant difference becomes possibly less significant the question arises will the mainframe be swamped by the cost/effectiveness of the mini-computer? There are several instances where installations are discarding their mainframe for a cluster of mini-computers, but this phenonomen is unlikely to happen to any great extent with the mainframe and mini-computer happily co-existing.

There are three possible options:-

(1) Mainframe Only.

The architecture of mainframes is going to change in a revolutionary manner in the next 10 or 20 years. The 'back-end' processor will become a standard component of the future mainframe. The processor will be able to perform hardware searching of the data base. Extra levels of storage will be available with mainframes. They will have an extra level of electronic storage which is slower but larger than direct main storage. Storage devices will be introduced that will contain much greater capacities than current disc storage devices. Storage capacities will be of the order of 10^{12} bits. The mainframe processors will need to search and process the data held on the several stages of storage device. Not only the hardware architecture but also the operating software will be developed on the theme of data access, data searching and data management.

(2) Minicomputers Only.

It could be a possible decision to have a minicomputer for each large scale project, but without any networking between machines would mean there could be no on-line integration of data between the projects. A concept gaining popularity is the building of a network based on the minicomputer. The machines would not necessarily be geographically remote but their individual functions would be separate. Each application would have its own machine while one or more machines would control access to the data bases and would perform the data base management functions. Similarly one or more machines would control the network. An implementation of the European D.B.M.S. package SIBAS already exists on such a network of minicomputers (Reference 10).

(3) Mainframe and Minicomputers in Same Network.

This is the most likely future outcome of an enterprise's expanding hardware requirements. The mainframe and minicomputer have opposing characteristics. The mainframe computer is suited to handling resources in the most efficient manner, while the minicomputer originating as it does from the process control area, is ideally suited to the handling of interrupts.

The minicomputer can provide a dedicated interactive service at a fraction of the price of the mainframe as it is ideally suited to either process control or, with the increasing commercial software facilities now being provided, it can support batch/on-line operation systems using a data base manager to access a localised data base. Several such machines can provide the basis for a computer service at a particular division of an enterprise. With improved network handling and the advent of distributed D.B.M.S., data base files can be accessed from different operating divisions in different locations. The various machines have the advantage of providing a resilient service enabling a service to be maintained even though one or possibly more minicomputers may have broken down.

The subject of distributed data bases will be discussed later, but an enterprise may still require a certain degree of centralisation of data on which to base the enterprise's complex management information systems. The need for mass storage of data, already described above, will still exist. The 90% of an organisation's data is probably only used 10% of the time, while the other 10% is used 90% of the time. The 10% used most frequently should be located where it is used while the 90% although infrequently accessed is probably no less important, providing the information on which possibly the future strategy of the enterprise is based. This mass of data must be readily accessible (on-line to the network). The large mainframe, designed with increased emphasis on efficiency in managing data resources must support the network's information resources supplying the enterprise with a full management information service to the divisions and head office. The information service will consist of summary, statistical and historic information supporting high-level 'English-like' enquiry facilities, statistical packages, explanatory design facilities, graphics design, strategic planning, historical surveys, market research, the list is endless, anything in the way of information that the modern manager may require. There may be fewer large machines than minicomputers but there contribution will be no less important. The minicomputer will relieve the larger machine of the 'bread-and-butter' work enabling the special mass data handling capability of the large machine to concentrate on providing the comprehensive information service.

What of the increasing emphasis on the use of microcomputers? Its influence will be in two spheres:-

(1) To provide the basis for advance in hardware architecture in so far as minicomputers and mainframes will be released from the concept of a single serial processor by large numbers of microprocessors functioning in parallel, increasing performance and flexibility of minicomputer and mainframe.

(2) As stand-alone devices their significance in all walks of life will be immeasurable. Apart from their use in machinery or domestic appliances, the use within the Management Service function will give new flexibility. Each programmer could be provided with his own microcomputer with COBOL compiler and testing aids to enable the development of his programs. Wherever a task is required to be done which does not need a large amount of data to be organised and accessed then the microcomputer can do the job. It is only when quantity of data to be handled increases significantly that microprocessors must be 'harnessed' together into minicomputer or mainframe architectures. Most tasks are untouched by computer and most tasks use only a small amount of data for their operation, this will be the growth market for the microcomputer.

The Management Services Manager must, when setting out with the data base plans for the present, consider carefully (although of course not accurately) the impact of today's developments on the Management Service world of the next 10 or 15 years. The manager's responsibility first and foremost of course is to the present.

7.6. THE USER'S REQUIREMENTS.

Reviewing the earlier part of this chapter, the Management Services Manager has directed his organisation to regard their data as a resource of the enterprise, irrespective of how it is processed. The manager has 'won over' top management to the idea, and has set up development teams to develop the first small projects based on this principle using a data base management system to

assist in the implementation. He has also directed the development teams to define the data and data relationships that exist in the enterprise's departments. Above all, the education exercise must be continued by the development teams within the user departments.

The Management Services Manager must keep abreast of the technological developments taking place and he must project the likely affects of these developments on the long term future within his own enterprise.

If the enterprise has a varied computer operation, with requirements for batch, on-line (both types as described in section 7.5.2.) and real time processing, what are the requirements of a D.B.M.S. to meet the needs of supporting such an environment?

Obviously, the 'laying of foundations', as described above, can only follow the manager's study of how to achieve the requirements of his enterprise, if indeed they are achievable. The clear answer is that they will be achievable to some degree. The manager must act as a salesman to user management extolling the virtues of a data base environment while being well aware that he must think conservatively about present-day implementations. He must prevent the implementation of data base applications that go beyond the current state of technology, too many applications in the past have become 'white elephants' for this reason. This is why when selling the concept to user management the D.P. executive must have analysed beforehand where technology is going to take his organisation in the future. The most important characteristics of the first implementations must be that performance (response time, throughput) is good and the systems are reliable. Each addition to the initial implementations must be looked at closely to see if performance and reliability will be affected. If this is the case, these additions should be suspended until such time that they become feasible.

How then does he decide what are his requirements from a D.B.M.S. and how does he decide if a D.B.M.S. meets these requirements? First he should consider what the Relational, CODASYL and GUIDE/SHARE proposals have to say.

7.6.1. The Approaches to Data Base Management.

The 3 approaches described in chapter 6, all have a contribution to make in the data base environment. They are often regarded as incompatible but each has its own contribution to make.

The Relational Approach.

This approach has 2 quite separate parts:-

(1) A way of defining data dependencies (normalisation) that exist so that all data dependencies are fully expressed within a grouping, whether that grouping is called a relation, an entity, a schema record or by any other terminology.

(2) The means of accessing such a grouping set without any spurious associations i.e. pointers, indices etc. These pointers and indices introduce accessing dependencies that pre-suppose how the data should be accessed.

The first point is an invaluable tool in defining data relationships within the schema independent of any D.B.M.S. or access method used.

The second point is exclusive to the concept of a relational data base management system. It implies an access method based on serial searching of the stored data. Such an accessing technique for large data bases without the use of parallel hardware searching is impracticable for performance reasons.

In operation systems, the access path is already defined and the chaining and/or indexing of data is practised in all commercially available data base packages. There are certain problems:-

- integrity problems may occur because of hardware/software failure (as described in Chapter 3);
- if the data base is restructured, pointers must be changed.

These are problems which must be carefully administered by the data base administrator, but they are necessary evils to permit adequate accessing performance.

The CODASYL Approach.

The concept of the SET as the basis of data structuring is simple but effective. It is the basic building block to develop any complex data structure. The only situation that is impossible to describe, is the many-to-many data relationship. Indeed it is to the credit of CODASYL that several data base packages are based on this approach.

The approach was first described in a report in 1971. In that report there appear several drawbacks. Notably, the proposed implementation is at a very low level. The Data Manipulation Language is semantically very simple but syntactically very complex. Semantically very simple means that the programmer must 'navigate' his way around the data structure he is accessing. He must know where he is, by using currency indicators. It makes the programming difficult and prone to error. Syntactically complex means that each verb has a complex set of options, in particular the FIND verb has 7 different formats, each one having different options.

The level of data independence is poor. The programmer has access to the data base key of a record whose value appears related to the storage of the record. The programmer has access to the AREA which is a storage concept.

The Data Description Languages for both Schema and Sub-schema are the same style as COBOL. Both languages could have been designed as a higher level language making the data descriptions easier to write and understand.

The idea that the CODASYL proposals and the Relational approach are incompatible is not necessarily true. In the previous section it was pointed out that normalisation rules, first defined within the Relational approach, are useable for any data system implementation even without using a data base approach. The Relational approach is geared to supporting ad hoc interrogation facilities and information systems where the access path cannot be pre-determined, while the CODASYL proposals with a pointer based approach is more suited to operation systems where access paths are pre-defined. It would be quite conceivable that with the advent of hardware searching both approaches could be included in the same D.B.M.S. provided that the pointers were stored separately from the data.

The Guide/Share Proposals.

These proposals are a set of requirements that users have specified to answer their data base needs, not a suggested means of implementation. In the sense that they are not implementation proposals, they are a lot less open to criticism, but more importantly they are USER requirements, and any computer system whether hardware or software which first and foremost supplies the answer to a user requirement has its objectives right. For too long in the age of 'bundled' software users have been provided with software that does not answer their needs, when the vendor's objective has been to sell more hardware. This failing has diminished with more market competition in software systems and 'plug-compatible' hardware, and with the diminishing cost of hardware, more emphasis has been placed by computer manufacturers on providing a total customer service.

The Guide/Share proposals are forward looking, attempting to direct base system developments.

The next chapter expands on these proposals, drawing on the details of the previous chapters of this thesis.

7.7. REFERENCES.

9. 'Principles of Data Base Management'.
James Martin.

10. Infotech State of the Art Report.
'On-line Data Bases'.

8. DATA BASE FACILITIES FOR OPERATION SYSTEMS.

8.1. INTRODUCTION.

8.1.1. Objectives and Strategy.

The basic objectives which must be sought by a user in his use of a D.B.M.S. should be those set out by the Guide/Share Committee. They are:-

- Data Independence;
- Data Relatability;
- Data Non-redundancy;
- Data Integrity;
- Security;
- Performance;
- Compatability.

Data Integrity and Security are absolutes in that every effort must be made to attain 100% data integrity and security. Data independence, relatability and non-redundancy are achieved tovarying degrees and each has a degrading effect on performance.

There are many aspects to each of the objectives with the emphasis of each objective varying from one part of the D.B.M.S. to another. The structure of this chapter covers the different parts of a ficticious D.B.M.S. and the attempt is made to show how each of the objectives may be met by each part of the D.B.M.S.

The parts of the D.B.M.S. described are:-

- The schema and data description language.
- The physical storage and data description language.
- The sub-schema and data description language.
- The data manipulation facilities.
- The data base manager.
- Other data base facilities.

The requirements are timeless, technology independent and environmentally independent.

The impact of technology will diminish the trade off situation between performance and the other objectives, with the capability to provide excellent access performance with very good data independence etc. With today's technology there is inevitably some trade off with very good data independence affecting the performance to some degree.

The environment which this fictitious D.B.M.S. should be capable of handling is as follows:-

(1) A mainframe computer system capable of being used in batch, on-line and transaction processing mode.

(2) Several stand-alone minicomputers, each one capable of operating in batch, on-line, transaction processing and proper real time (process control) modes. Each minicomputer may be connected to the mainframe system but only to be used for off-line bulk data transfer.

(3) The use of intelligent terminal systems used in both off-line and on-line mode. The terminal systems may be connected to either a minicomputer or mainframe.

8.1.2. An Example Application.

An application is used to enable the description of the facilities of the D.B.M.S. package, taking into account the type of machine environment described above.

The application is that described previously in section 3.5.7. The unnormalised business relationship is transformed into the 4 third normal form relations, as follows:-

- PRODUCTS P (PN, PD, QL, QOH)
- PRODUCT ORDERS PO (PN, ON, Q)
- ORDERS O (ON, CN)
- CUSTOMERS C (CN, CNA, CA, CP, X)

The attributes are as follows:-

PN - part number (primary key of the 'PRODUCT' entity);
PD - part description;
QL - quantity level;
QOH - quantity on hand;
Q - quantity ordered;

ON - order number (part of the primary key of the 'PRODUCT ORDER' entity);

CN - customer number (primary key of the 'CUSTOMER' entity);

CNA - customer name;

CA - customer address;

CP - customer priority;

X - export order.

QUANTITY LEVEL, CUSTOMER NAME and CUSTOMER ADDRESS are attributes that have been added to the original BUSINESS entity.

The diagram of the schema for this application is as follows:-

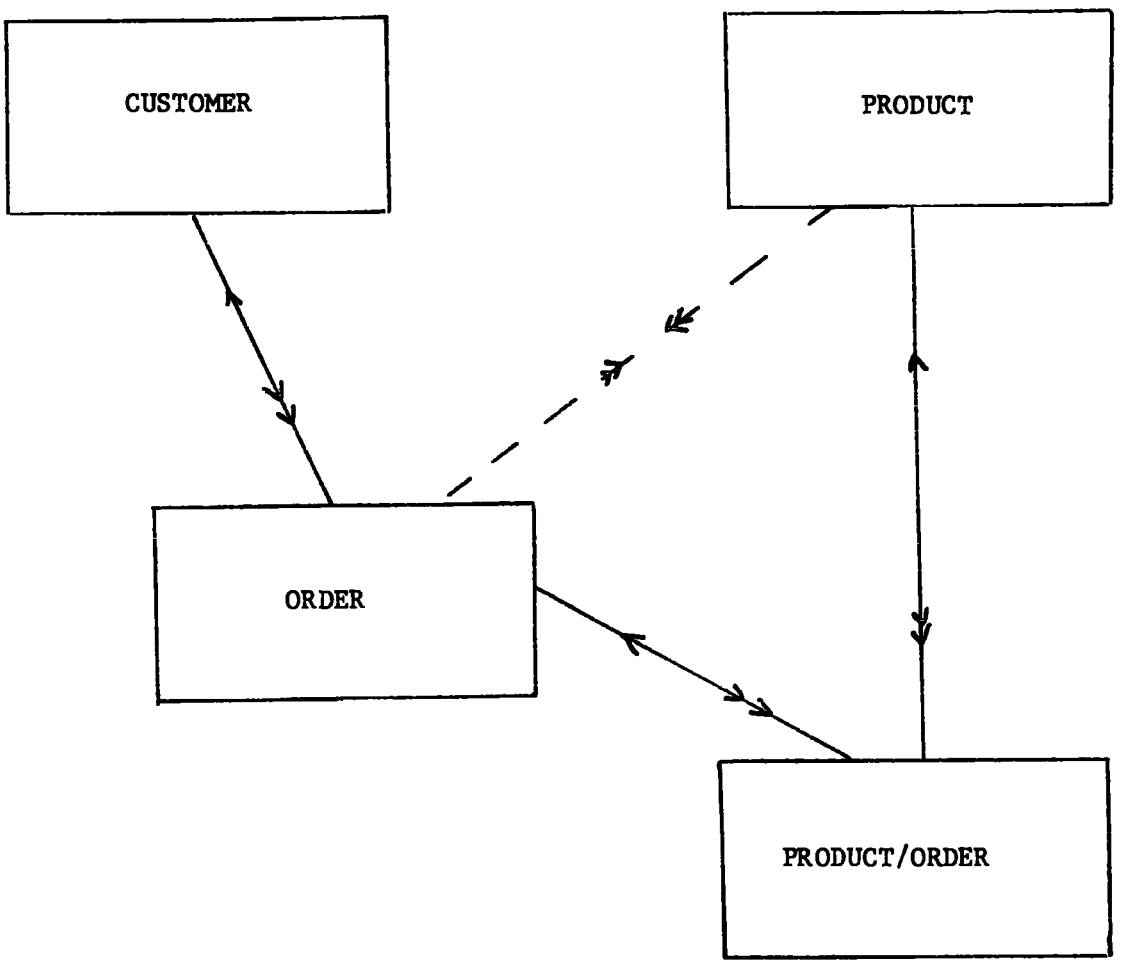


FIGURE 8.1. APPLICATION SCHEMA.

The relationship between PRODUCT and ORDER is an 'm:n' relationship, but with normalisation this relationship is replaced with a new entity, the PRODUCT/ORDER entity. The creation of this new entity is a natural result of normalisation. The CODASYL approach recommends that the same result be achieved by replacing the original set of ORDER record as owner to the member record PRODUCT with 2 sets.

It is worth listing typical transactions that are likely to be performed on the data base:-

For a product -
- new product included;
- old product deleted;
- alter the threshold quantity level for a product;
- a quantity of a product is brought into store.

For a customer -
- new customer;
- change of customer address;
- change of customer priority;
- production of customer order forms.

For an order-
- new order received;
- alteration to the details;
- production of picking list for warehouse;
- production of invoice;
- deletion of order when payment sent.

These are some of the transactions that could possibly be performed on the data base.

The system has 4 entities describing 3 subjects:-
- the customer;
- the product;
- the order.

Following on the idea of having subject data files rather than application data files, there are 3 data files required, one for each of the above subjects. This application emphasises the point that even when a small application is being implemented quite a large number of subject files are required.

The fields making up the files are as follows:-

Customer file -
- customer number (key field);
- customer name;
- customer address;
- customer priority;
- exporting customer.

Product file -
- part number (key field);
- part description;
- quantity level;
- quantity on hand.

Orders file -
- order number (key field);
- customer number;
- part number;)
- quantity of part ordered.) These 2 fields will occur a variable number of times.

There is a degree of data redundancy with the CUSTOMER NUMBER and PART NUMBER present in 2 files. For performance reasons this is far better than having no data redundancy with field pointers between files. The redundancy could be overcome by storing all the data in one file in the same manner as the original unnormalised BUSINESS entity described in section 3.5.7. The single data file would mean that the data would be tied to the application by the way the data is stored, making future integration very difficult and eventually causing much greater data redundancy.

8.2. DATA DESCRIPTION.

The following sections describe a data description language with the facilities that today's user may require. The facilities describe schema, sub-schema and storage structures in turn. The facilities initially described may not be final and may be added to during this and later chapters. The structure of data at 3 levels (as described in Appendix 1) is used as the basis of the data structure definition.

The basic characteristics of the language are:-

(1) It is independent of all other languages. Although statements may differ from level to level, the language should be able to describe all 3 levels of data description.

(2) The language should be easy to understand. The structure that is being described should be definable by the way the language statements are laid out.

(3) The language should be extensible with new primary functions easily added and extensive provision for users' own procedures.

A full layout of statements for both the data description language and data manipulation language is provided in Appendix 3. Throughout this and later chapters the example application described above may be used to illustrate a point.

8.3. THE SCHEMA.

There may be several schemas defined by an enterprise. This is because there may be no relationship between certain entity types and there is therefore no need to go to any greater degree of integration than is necessary, remembering a schema describes a 'family' of entities that are related . A schema must have a unique name. The first statement to describe a schema should be:-

SCHEMA schema-name;

N.B. All words spelt in capital letters are language reserved words. Each statement is terminated by a semi-colon.

8.3.1. The Entity.

The basic building block of the schema is the entity. Each entity type must be named with a unique name. The name should be globally unique, that is to say that an entity in another schema cannot have the same name.

The entity is named as follows:-

ENTITY entity-name;

8.3.2. The Attribute.

Each entity is describable by one or more attributes. Each attribute must have a unique name within the whole schema and should be named as follows:-

ATTRIBUTE attribute-name;

The key words SCHEMA, ENTITY and ATTRIBUTE are able to be used in a 3-letter shortened form i.e. SCH, ENT, ATT.

Every attribute is uniquely defined by the following expression:-

entity-name. attribute-name

A single attribute can be present in different entities e.g. in the example the attribute CUSTNO (customer number) is present in entity ORDER and entity CUSTOMER. The same attribute is used to describe 2 different entities.

The attribute requires to be described in terms of its length and class. The COBOL PICTURE description is ideally suited for this purpose:-

9 - indicates a numeric character;
A - alphabetic or space;
X - alphanumeric.

8.3.3. Attribute Validation.

The ORDER NUMBER attribute (ORDNO) as described in section 3.5.7. would be 99A or 9(2)A.

Range checks should be available to check for values outside a valid range or conversely inside an invalid range, i.e.

VALID RANGE = 1 - 4;
INVALID RANGE = A - H;

Similarly it should be possible to define valid/in-valid values:-

VALID VALUE = C␣D␣Z ;
INVALID VALUE = G␣W;

This facility should permit up to 5 valid or invalid values to be specified.

If the list of values either valid or invalid is longer, then such a set of values should be set up in a list by the D.B.A. and referred to by the list name, thus:-

VALID LIST list-name;
INVALID LIST list-name;

The list may contain values and/or ranges e.g.:-

AA␣CZ␣EX␣16-32␣37␣94

The validity of an attribute value may be dependent on the relationship between several sets i.e.

VALID RESTRICTION LIST list-name$_1$, list-name$_2$.....list-name$_n$;

RESTRICTION means that the resultant list of valid (or invalid) values are those that are present in all the lists e.g.

list 1 : X␣Y␣Z␣W␣14-18␣A
list 2 : Z␣B␣16-18
list 3 : Y␣Z␣C␣15-18

The RESTRICTED list would be: Z 16-18

COMPOSITION is the converse of RESTRICTION and includes all values that are not common to the lists, this gives:-

X␣Y␣W␣B␣C␣A␣14-15

JOIN means that all values within the list are included, thus:-

X␣Y␣Z␣W␣14-18␣A␣B␣C

Similarly, attribute values from a single run-unit can be specified to be in a specific sequence, thus:

VALID SEQUENCE LIST list-name;

In the above examples INVALID may be used instead of VALID.

There should be special procedural checking i.e. numeric digits are related to others in the same attribute value. The most common example is the date format.

Extra facilities are required beyond those already mentioned, they are:-

(1) The ability to perform such validity checks as already described, on parts of attribute values.

(2) The ability to relate part of an attribute value with another part of the same attribute's value.

The facility is required to use a procedure as well as other validation methods, thus for date checking:-

```
(1)  ORDER.DATE PIC +9(6);
(2)  ORDER.DATE (3,2) VALID RANGE = 1-12;
(3)  VALIDATION PROCEDURE DAYS ORDER.DATE (1,2), ORDER.DATE (3,2), ORDER.DATE (5,2);
```

The validation procedure used would be expressed as:-

```
(4)  PROCEDURE DAYS;
(5)  TABLE MAX-DAYS;
(6)  IF %A > MAX-DAYS (%B)
(7)  INVALID
(8)  ELSE IF %B NOT = 2
(9)      GO TO PROCEND
(10) ELSE IF %A NOT = 29
(11)     GO TO PROCEND
(12) ELSE IF %C = 00
(13)     GO TO PROCEND
(14) ELSE W1  = %C/4
(15) IF REMAINDER = 0
(16)     GO TO PROCEND
(17) ELSE INVALID;
(18) PROCEND;
```

Some notes on the above example are as follows:-

At step (1) the '+' sign indicates only a positive value is permissible. In the second step, the (3,2) identifies the part of the attribute value that is being validated '3' indicates the starting character and '2' indicates the number of characters to be validated.

The procedure, in this case DAYS, uses 3 parameters. The table MAX-DAYS gives the total number of days in each month. The major part of the procedure is dealing with checking for a leap year.

More details concerning procedures and the handling of invalid conditions will be dealt with later. The use of procedures permit the user to extend his facilities to the level he requires. This date procedure would be called for any date attribute, but this type of procedure should really be provided by the D.B.M.S. as a system procedure. The availability for the user to develop his own procedures permits his own specific requirements to be serviced.

8.3.4. Coding.

There should be automatic compaction between the schema attribute value and the stored field value. This provides 2 purposes:-

(1) Less storage space is required.
(2) Privacy is enhanced with coded data.

Coding facilities should be available at all three levels of data description. Although the way such a coding mechanisms would be intended to work would differ from level to level the facilities would be the same and the user could make use of such mechanisms, or not, as he desired. The emphasis should differ from level to level as follows:-

(1) At the schema level the emphasis should be placed on the coding of data to give the particular attribute definition.

(2) At the sub-schema level, to enable a particular user's required use of a data element to change from the default real world definition of the attribute in the schema to the user's own specialised definition.

(3) At the physical level, to emphasise the storage requirements of a data field and so to implement the most effective compaction techniques.

At the schema level, the idea of an attribute having definition was first defined in the section on data relationships (section 2.2.3.). If the'height of a person' has a stored value of '6', does that mean 6 inches, 6 feet or what? A value should only be stored in one definition probably the 'highest common denominator' in this case inches. The most usual use of the stored field as

an attribute value will probably be feet and inches if the attribute is 'the height of a person'. The attribute must have associated with it a coding procedure to perform the conversion from inches to feet and inches, thus:-

CODING PROCEDURE procedure-name;

The effect of the procedure would be to convert the stored data value into a 2-value attribute with the last 2 digits of the attribute defined as inches and the remainder as feet.

The D.B.M.S. should itself provide general procedures such as the above procedure, but the user should be able to write his own and he should be given facilities to do so. Details of procedures will be described later.

Another method of coding, if there are only a limited number of possible attribute values, is to use a table rather than a procedure:-

CODING TABLE table-name;

8.3.5. Virtual Attributes.

These attributes are not mapped to a physically stored field but are the results of computation based on the values of other attributes. The format of the instruction is:-

VIRTUAL PROCEDURE procedure-name;

8.3.6. Attribute and Entity Termination.

The attribute and entity descriptors should be terminated by an END statement, as follows:-

ATT END;
ENT END;

All attribute descriptions must terminate before the next attribute description starts and all attribute descriptions must be fully nested within its entity description.

8.3.7. Primary and Secondary Keys.

Following all the attribute descriptions but before the termination of the entity description the primary key must be identified as well as any secondary keys that are required. There must be at least one primary key and any number of secondary keys.

The keys can consist of concatenated attributes or overlapping attributes if they are character strings. The keys are defined as follows:-

```
PRIMARY KEY key-name = entity-name.attribute-name (x,y):
                       entity-name.attribute-name (x,y):...;
```

```
SECONDARY KEY key-name = entity-name.attribute-name (x,y):...;
```

as has already been described in the validation procedure above:-

x - indicates the character position within the attribute that the key starts;

y - indicates the number of characters;

The x and y values are optional, if they are missing the whole attribute is used in the key. The concatenation is also optional and is indicated by the presence of a colon.

The secondary key can also be derived by using a procedure:-

```
SECONDARY KEY key-name = PROCEDURE procedure-name;
```

Normally individual secondary key values provide the basis for an entity belonging to that particular index. It may be required for membership of a secondary index to be based on a range of values. The facility is provided by including a RANGE instruction with the secondary key instruction as follows:-

```
SECONDARY KEY key-name = entity-name.attribute-name
VALUE RANGE FROM x BY y TO Z;
```

```
If x = 1
   y = 3
   Z = 9
```

then an index is provided if the key name has a value from 1 to 3, another index for values 4 to 6 and finally an index for values 7 to 9. Any number of these ranges may be included for a single key. If all possible values were not provided for in the ranges there would be an index for these individual value.

If there are several ranges for a single key, the ranges can be grouped into a table.

The ranges will not relate solely to numeric values, character string ranges should be available as follows:-

VALUE RANGE FROM AAA BY 2 TO AAD;

This means that AAA, AAB are included in the same index and AAC, AAD are together in a different index. Only the BY qualifier must be numeric. The range can straddle alphabetic, alphanumeric and numeric strings. This statement is machine dependent because with some machines numerics have a higher representation value than alphabetics, and for other, alphabetics have a greater value than numerics. Ranges may be exclusive to a value class i.e. NUMERIC, ALPHABETIC or ALPHANUMERIC e.g.

NUMERIC VALUE RANGE FROM 1 BY 5 TO 100;

8.3.8. Procedures.

Already several instructions in the schema have been described that use procedures.

The basic method of calling a procedure is:-

instruction PROCEDURE procedure-name $param_1$, $param_2$,...;

The parameter string is optional and to save repetition they have not been included in instruction definitions but any procedure called may use parameters.

The procedure body has the initial header:-

PROCEDURE procedure-name; (PROCEDURE can be shortened to PROC).

and the procedure body ends with:-

PROCEDURE END; (this can be shortened to PROCEND)

The type of instructions and operators are as follows:-

Arithmetic Operators - the normal arithmetic operators can be used (+,-,*,/). The operands will be 2 attribute or literal values.

Comparison Operators - the normal 'less than', 'greater than', 'equal to' (<, >, =) are permissable.

Logigal Operators - AND, OR, NOT are all permissible.

The use of brackets permit operations to be nested.

'IF, ELSE' Commands.

The 'IF, ELSE' conditional statement permits decisions within the procedure to take place. The 'IF,ELSE' clauses can be nested.

The FOR statement.

The FOR statement specifies the domain of the procedure i.e. it defines which attribute values are to be processed. There are 2 forms of this statement:-

FOR ANY statement.

This ANY qualifier states that if there is a single occurrence within the domain specified where the procedural condition holds, then the resultant action takes place.

'FOR ALL' statement.

The ALL qualifier states that all occurrences within the domain specified must meet the procedural condition before the resultant action takes place. If no conditional statement, is included it is possible to use this statement to perform a 'blanket' operation on all domain entries.

The use of the commands will become clear as examples are given.

Parameter Substitution.

Positional parameter substitution is used so that within the body of the procedure %A represents the first parameter, %B the second parameter etc.

8.3.9. Integrity.

Integrity at the schema level is primarily associated with ensuring that data to be stored on the data base meets the validation requirements. The normalised structuring of entities ensures that the attributes of an entity are fully dependent on the entity identifying attribute. This means that the attributes, other than the identifying attribute, within an occurrence of an entity relate only to other attributes within that occurrence. The area of integrity only exists within that occurrence. At the 'real world' description data can be regarded as 'packets' of related data. Certain integrity checks are required to be made on entity identifier occurrences e.g. is this new order

number unique? does this customer number exist? These checks however are related to the way the data is being processed rather than relating to any meaningful relationship which is independent of processing. If a new order is being raised then it is important that there is not an order number already on the data base with the same value. If the quantity of a product on an existing order requires changing, the order number must be checked to already exist. Two opposing requirements based on the same entity identifier. This type of integrity checking should be performed at the sub-schema level.

In the business example there is one meaningful relationship which must be checked at the schema level, i.e. the non-directed relationship between PRODUCT QUANTITY-ON-HAND and PRODUCT RE-ORDERING LEVEL.

The format of the integrity instruction is:-

INTEGRITY PROCEDURE procedure-name;

An example describing this quantity checking procedure is illustrated later in the chapter.

8.3.10. Privacy.

A great deal of data privacy worries are removed in operation systems by the fact that the specification of the data required removes any danger of the wrong type of data being accessed. It would be impossible while using a production control system for a terminal operator to access personnel information if the D.B.A. had set up the sub-schema correctly so that access to such information is made impossible.

Similar to integrity it is unlikely that privacy restraints will be placed at the schema level in an operation system, they are much more likely to be at the sub-schema level where the context of data use is described.

Should a privacy procedure be required at the schema level, it should be situated at the same place within the entity description as the integrity procedure, thus:-

PRIVACY PROCEDURE procedure-name;

8.3.11. Entity Relationships.

After all the entities in a schema have been described there is the requirement to describe how the entities relate.

The relationships to be described are as follows:-

RELATION;

relationship-name? entity-name = entity-name; (1-to-1 relationship)

relationship-name? entity-name > entity-name; (1-to-many relationship)

relationship-name? entity-name < > entity-name; (many-to-many relationship)

RELATION END;

As an example; to describe the relationship between a customer and his many possible orders:-

CUSTORD? CUSTOMER > ORDER;

To enable multiple relationships to be described, the same entities are described in the same way with a different relationship name. In the example system the entity PRODUCT-ORDER may exist in 1 of 2 relationships a SATISFIED relationship meaning the quantity of the product ordered is available and a SUSPENDED relationship means thatthe product on order is not in stock. The 2 relationships would be described in the schema as follows:-

SATISFIED? ORDER > PRODUCT-ORDER;

SUSPENDED? ORDER > PRODUCT-ORDER;

To fully identify an attribute of an entity belonging to a particular relationship, the following qualifications are required:-

relationship-name? entity-name.attribute-name (x,y)

It is possible to build up a structure of relationships. In a hierarchically structured schema an entity at level 3 in the hierarchy may be related by a multiple relationship to an entity at level 2 in the hierarchy which is itself related in a multiple relationship to an entity at level 1.

An attribute would require to be identified as follows:-

relationship-name$_1$? relationship-name$_2$? entity-name.attribute-name (x,y)

The schema diagram for the business example could be re-drawn to show the multiple relationships.

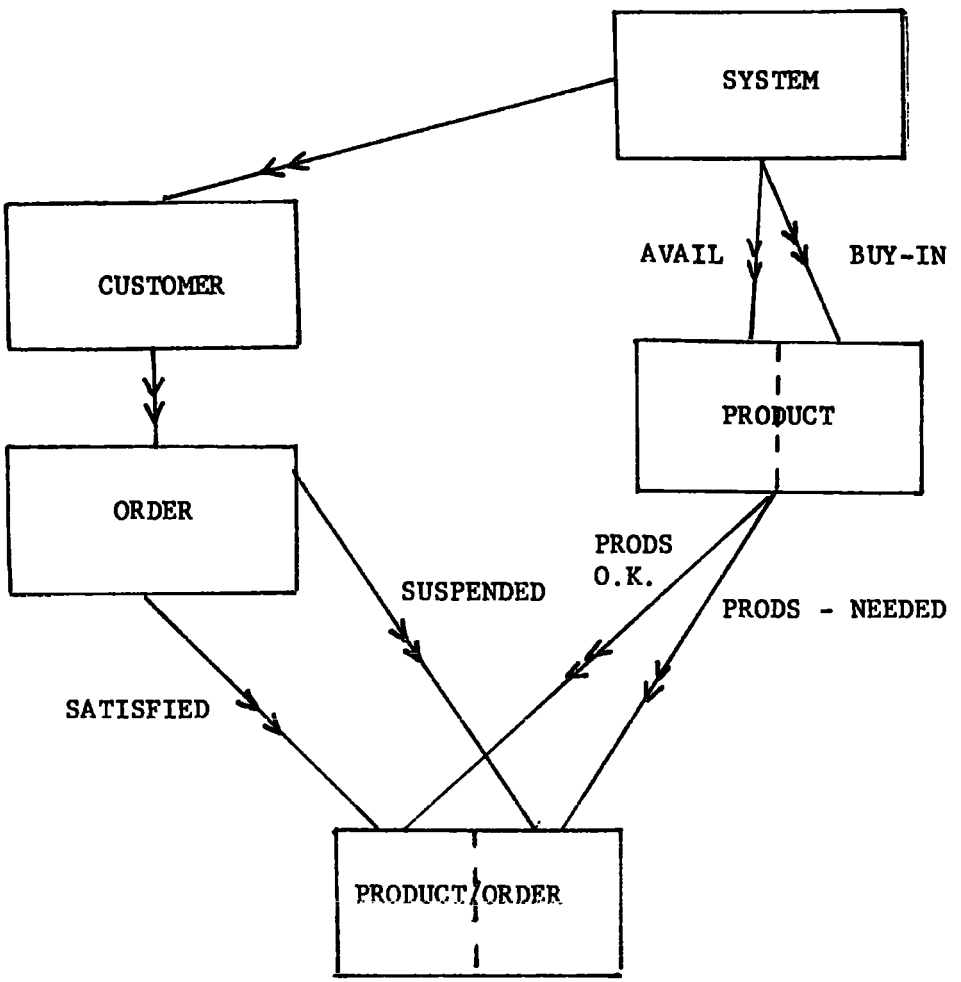


FIGURE 8.2. MULTIPLE RELATIONSHIPS IN SCHEMA.

All entities in a schema must have a controlling entity 'THE SYSTEM'. This concept was introduced in the CODASYL proposals.

'THE SYSTEM' entity does not exist but is a means of describing the 2 product relationships 'AVAIL' and 'BUY-IN'.

In the schema the relationships would be described as:-

AVAIL? SYSTEM > PRODUCT;

BUY-IN? SYSTEM > PRODUCT;

To enable to describe the completeness or otherwise of entities belonging to one of several relationships, a virtual data element is available to any run-unit. The data element has an identical name to that of the relationship.

This data element can take one of three values:-

-'0'- there are no subordinate entities belonging to this relationship;
-'1'- some subordinate entities belong to this relationship;
-'2'- all subordinate entities belong to this relationship.

This means that in a '1-to-n' relationship, this flag has a value of '2' if all 'n' entities belong to the corresponding relationship e.g. if all products for a single order can be satisfied then 'SATISFIED = 2' for that ORDER entity.

If within a schema integrity procedure there is a requirement to define that a particular entity occurrence should belong to a particular relationship, the following instruction would be used:-

```
RELATIONSHIP SYSTEM = BUY-IN;
```

The integrityprocedure would be called within the PRODUCT entity description.

It is possible to define multi-level relationships as follows:-

```
BUY-IN? PRODS-OK? PROD-ORD.QUANTITY
```

This defines the 'quantity ordered' for a product which was originally available (i.e. when the order was placed stocks were high enough to enable allocation) but since then stocks have dwindled below BUY-IN levels.

Because there are 2 relationships between the SYSTEM and PRODUCT and 2 relationships between the PRODUCT and PRODUCT/ORDER there could be a maximum of 4 combinations. However it may not automatically follow that all 4 combinations are valid. To restrict access to any combination the following set of instructions are required:-

```
RESTRICTION;
relationship-name_x? relationship-name_y? entity-name;
RESTRICTION END;
```

Any combination not restricted in this way would be possible. This controls the indiscriminate setting up of relationships which as combinations may be in 'real world' terms impossible to exist.

Loops.

It is possible to describe the loop structure which relates an entity type to itself as in the PART structure, hence:-

SET-UP? PART> PART;

8.3.12. Homogeneous Structures.

The LOOP relationship is identifying a homogeneous tree structure (i.e. different entity occurrences of the same entity type related as a tree structure).

The ORGANISATION sub-schema which illustrates the construct idea in figure 2.17 is a good example of a LOOP in the real world. Within the PROJECT there is a TEAM of EMPLOYEES and a PROJECT MANAGER. Within the DEPARTMENT there are PERSONNEL and MANAGEMENT groupings. There are differing relationships between employees, differing hierarchies for TASK responsibility and PEOPLE responsibility. Nevertheless all relationships are homogeneous, there is only one entity set EMPLOYEE. The need is to provide a mechanism to give a subset of EMPLOYEE occurrences (identified by the primary key EMPLOYEE NUMBER) by supplying an access path which uses different secondary indices.

Within the schema, the secondary key definition would require extension as follows:-

```
SECONDARY KEY key-name-x = key-name-a WITHIN
                           key-name-b WITHIN...;
```

In the ORGANISATION example the PROJECT MANAGER EMPLOYEE occurrence would be defined in the schema as:-

```
SECONDARY KEY PROJECT-MANAGER = EMPLOYEE-STATUS WITHIN PROJECT;
```

Each EMPLOYEE entity would have EMPLOYEE-STATUS and PROJECT attributes and the attributes would require to have already been defined as secondary keys within their own right earlier within the schema description.

This relationship is really a plex homogeneous structure because the 'PROJECT MANAGER EMPLOYEE' exists within the DEPARTMENT relationship. A different secondary key derived from a different attribute, the DEPARTMENT.

Not all homogeneous relationships are so naturally defined at the schema level i.e. the relationship is not definable by certain attribute values which already exist as a natural description of the entity occurrence. The bill-of-materials relationship where different parts relate differently to other parts dependent on the structure being built. In such a structure a different facility is required. A 'STATUS-LEVEL' must be defined as a secondary key for the entity, hence:-

SECONDARY KEY key-name = STATUS-LEVEL;

The values that the secondary key can take will be described under the section describing the Command Language later in this chapter.

An integrity procedure could be employed at the schema level to check that there is only one employee of 'PROJECT-MANAGER STATUS' belonging to the particular project. Every time an EMPLOYEE entity is written or updated the integrity procedure would be involved to check the STATUS attribute and if necessary signal an integrity failure.

The 'cycle' can similarly be defined as follows:-

A? x > y;
B? y > Z;
C? Z > x;

8.3.13. Schema End.

The schema description is terminated by the instruction:-

SCHEMA END;

8.3.14. Conclusion.

The schema defines the pure attribute data identities on which is based the mapping to logical and physical data. The main emphasis is on the validity of attribute values with less emphasis placed on the integrity of related data because dependencies are implicit with the data structured in normalised form

and with less emphasis on access control because this largely depends on the context of the data, where it is going and how it is to be used all of which are more closely related to the concept of the sub-schema.

8.4. THE SUB-SCHEMA.

The purpose of the sub-schema is to describe the possibly complex data structures that require manipulation, to define integrity and access control mechanisms based on how the data is used all within the definition of how the data is related at the schema level. The structuring at the schema is only 2-level, the attribute within the entity, but the sub-schema structuring should be able to be expressed at any number of levels.

One restriction which must be placed on the sub-schema is that it is based on one and only one schema.

The first statement in the sub-schema should identify the sub-schema:-

SUB-SCHEMA sub-schema-name;

8.4.1. The File.

In a batch environment and in any requirement for access to a large amount of data there is a need to provide a facility to collect all the data together before processing. Such a facility would be a logical file containing a variable number of record occurrences. The file is described as follows:-

FILE file-name;

The file entry is optional and would not usually be used in an on-line environment.

8.4.2. The Record.

The record describes the total data structure of the sub-schema:-

RECORD record-name;

8.4.3. The Data Item.

This is the smallest structural unit in the sub-schema. Each item has associated with it a name. The mapping must be defined to enable the data item to be materialised from the schema description. The simplest mapping is where there is a '1-to-1' correspondence between the data attribute and the data item, this is described as:-

ITEM item-name = entity-name.attribute-name

The item can be called by the same name as the attribute-name or it can be called a different name. Items can be constructed from a part of an attribute or by concatenation of whole and/or part attributes. This is expressed as follows:-

```
ITEM item-name = entity-name.attribute-name (x,y):
                 entity-name.attribute-name (x,y)
```

In addition, the item mapping can be further qualified by associating a relationship-name with the entity-name should such a relationship exist, this gives:-

```
ITEM item-name = relationship-name? entity-name.attribute-name (x,y):
                 relationship-name? entity-name.attribute-name (x,y)
```

The picture description is the same as the schema picture description with the addition that an item can be described as a valid internal machine representation (e.g. binary, packed decimal etc.) to enable efficient computation by the application program.

As an alternative to mapping from attributes, the data element may be virtual. The data element may be derived by procedure in an identical manner to the virtual procedure described in the schema.

It is possible that a particular coding change may be required to convert the schema value to an application related value. The coding is performed by a procedure associated with the item. The format of the statement is the same as the equivalent schema statement.

It should be possible to redefine the initial definition of the item equivalent to the method of redefinition in COBOL.

8.4.4. The Data Aggregate.

The data aggregate represents a named grouping of data items. The aggregate is named and ended in the following way:-

```
AGGREGATE aggregate-name;
AGGREGATE END;
```

AGG can be used as a shortened form for AGGREGATE.

The facility should be provided to map to the schema attributes at this level, then the items sub-ordinate to the aggregate would have an implicit mapping by virtue of the size of the item's PICTURE.

Data aggregates can also be redefined.

8.4.5. The Vector.

A vector is a collection of items and/or aggregates that occur more than once, there should be no need to supply a maximum value on the number of occurrences. The vector statement is:-

VECTOR vector-name;

the end of the vector is signalled by:-

VECTOR END:

It is possible to achieve considerable structural definition using items and item groupings.

8.4.6. Access Control

Access control is more effective at the sub-schema level than at the schema level for operation systems because the sub-schema is dealing with the accessing requirements of the requestor; who is the user? from which terminal is he accessing the data base? how does the requestor intend to use the data obtained? (The requestor is anyone or anything wishing to obtain data from the data base.)

The first point that has already been mentioned is that the sub-schema, providing, as it does, a limited picture of the data base, is the most significant access control mechanism available, restricting the attribute and entity types that are available to the requestor.

Before describing access control further, the term data element is meant to describe any collection of data in the sub-schema from a data item to a file.

The first access control mechanism permits or restricts access of a terminal to a data element. The mechanism to enable this control uses a terminal list:-

INCLUDE TERMINAL LIST list-name;
EXCLUDE TERMINAL LIST list-name;

Obviously whether the permitted list or restricted list of terminals is the shorter determines whether the INCLUDE or EXCLUDE statement is used. The terminal list may consist of terminal numbers or possibly, more likely, a list of terminal names to permit the list to be as independent as possible from the physical terminal characteristics.

The identification as to which data element the access control applies, is implied by the position of the statement within the sub-schema's nesting structure.

A similar control can be applied to user identifiers even for the same data element which has terminal restrictions placed upon it. The structure of the statement is:-

INCLUDE USER LIST list-name;
EXCLUDE USER LIST list-name;

The separation of user and terminal access control is useful to permit different users to use the same terminal.

A further qualification is required for terminal and user access control, the need to define what mode of access is being permitted or restricted. To READ, WRITE (UPDATE), DELETE or OUTPUT (to the outside world to printer or VDU's). There are 2 types of deletion, logical deletion which deletes the sub-schema's reference to a particular data element occurrence and the physical deletion which deletes the data field(s) from the stored data base file. The read, write, logical delete and physical delete are increasing degrees of access levels, if for instance a terminal is given the capability to physically delete a data element the other types of access are implied whether as a restriction or as a permission. The output access authorisation is independent of the other authorisation levels. There are then a maximum of 2 qualifications, one for the level of data element access and the other to indicate whether output is permitted. Expression of the qualification may be by keywords or by assigning a value to a data-name. The value is a 2 digit value:-

- the first digit is either '0' or '1', the value '1' indicates the presence of the OUTPUT qualification;
- the second digit is in the range '1' to '4' with the value '4' indicating the physical delete level and '1' the read-only level.

As an example:-

```
INCLUDE OUTPUT DELETE TERMINAL LIST list-name;
or INCLUDE data-name TERMINAL LIST list-name;
```

Where data-name has been set previously to the value '14'.

The meaning of INCLUDE to the qualification levels is to say permit output, physical and logical deletion, writing and read of this data element to these terminals. The same qualification to the exclude implies the physical deletion and output are not permitted at these terminals but logical deletion, writing and reading is permitted. If there is no qualification, everything is permitted or conversely everything is restricted to those terminals in the list. The keyword REMOVE is used to represent logical deletion.

It is also possible to state for any data element:-

```
EXCLUDE OUTPUT DELETE;
```

this is interpreted without terminal or user qualification that this data element is not to be output or physically deleted when accessing the data base using this sub-schema.

It may be unsatisfactory to restrict access to a data element without regarding its value or its relationship to other data elements within the same sub-schema. The access control may be actioned by procedure as previously described for attribute value validation in the schema. The same parameter facilities should be provided and the various INCLUDE/EXCLUDE statements may be used within the procedure.

Access to a SALARY data item must be restricted to the PERSONNEL MANAGER if the SALARY is greater than '£8000' and the EMPLOYEE STATUS CODE is less than '03' (manager). A procedure is called under the SALARY data item, thus:-

```
ITEM SALARY = PERSON.SALARY PIC.....etc.
PRIVACY PROCEDURE CHECK-USER PERSONNEL, 8000.00, 03;
        - - - - - - - - - - - - - - - - -

PROCEDURE CHECK-USER;
   IF DATA> %.B AND
    EMPLOYEE-STATUS < %.C
     INCLUDE OUTPUT WRITE USER LIST %.A;
PROC END;
```

This procedure would give output, read and write facilities to which ever user names were in the PERSONNEL list. The facilities given would be performed on item SALARY if the value of this item is greater than '8000' and the STATUS of the EMPLOYEE with that SALARY is higher than '03' ('01' highest). The word DATA represent the data element being referenced, in this case SALARY (i.e. 'SALARY' is an implied parameter to the procedure. The dashed line above indicates the procedure body is not within the sub-schema description, part of which appears above the dashed line).

There is the need to restrict access to certain values of data, if a different data element in the same sub-schema has a certain value. In the BUSINESS example certain products may not be available to a particular customer so that on the customer order form only those 'products' that may be ordered by that 'customer' will appear on the order form.

One solution to this type of problem is to use the idea of authorisation and category levels. The data element (CUSTOMER) which wants to obtain a list of another data element (PRODUCTS) must have an authorisation value for the customer that is greater or equal to the category level of the product to enable that product to be included on the list of products for that customer. For each new product stored in the data base a certain level must be appropriated, similarly a new customer would be given a level value. Within the data element description within the sub-schema a data name is defined to hold the access value, hence:-

```
ITEM CUST-NO
  ACCESS LEVEL CUST-LEVEL;
```

The program or process using this sub-schema would set the value before writing the new CUSTOMER data element to the data base. This technique is simply partitioning all data element occurrences into one of several levels. This technique could become impossible to operate should other data element types demand different accessing restrictions on the PRODUCT data element.

A better solution would be to associate with every customer occurrence a list of product ranges or even a list of single products that may or may

not be accessed. An example of this type of access control would be as follows:-

```
ITEM CUST-NO
   ACCESS TABLE PRODUCTS-RESTRICT;
- - - - - - - - - - - - - - - - - - -
TABLE PRODUCTS-RESTRICT;
   customer-x-value EXCLUDE VALUE LIST PRODUCTS-FOR-X;
                    ¦
                    ¦
TABLE END;
```

A further refinement might be to include access qualifiers against the value lists within the table, hence:-

```
customer-x-value EXCLUDE OUTPUT VALUE LIST PRODUCTS-FOR-X;
```

For any customer occurrence there may be several different product restriction lists, one for a different type of access. The thoughts of a user wishing to implement these access control facilities to any great extent involving more than a handful of data elements in the whole data base would be a nightmare to the data base administrator and the effects on performance with all this checking could be horrific, however if used sparingly and in the right context they could be useful facilities provided their use was strictly controlled. The terminal and user access control facilities described earlier would probably be much more valuable and much less of an overhead to implement. The terminal restrictions need not only be for transaction processing terminals but for any type of terminal including remote batch terminals.

8.4.7. Integrity.

What might be called 'pure' integrity at the schema level has already been described, however there is a greater amount of integrity checking based on the way data is being processed. As at the schema level a procedure is used to perform integrity checking. The same INTEGRITY instruction is used and the same parameter facilities are available to be used with the procedure.

Using the customer/order application as an example, the following procedure can be used to check the uniqueness of a new order, a new customer or a new product:-

```
PROCEDURE CHECK-UNIQUENESS;
   FOR ANY %A
      IF %A (CURRENT) = %A
         FAIL %B;
PROCEND;
```

The procedure would be called once for each of the 3 data elements:-

```
INTEGRITY PROCEDURE CHECK-UNIQUENESS ORD-NO, 1;
INTEGRITY PROCEDURE CHECK-UNIQUENESS CUST-NO, 2;
INTEGRITY PROCEDURE CHECK-UNIQUENESS PROD-NO, 3;
```

Parameter B provides a number indicating the reply to the particular integrity failure. The reply would be passed back to the run-unit. All data elements used in the integrity procedure must be defined in the sub-schema.

Further characteristics of the sub-schema will be defined in the section describing the data base command language.

8.4.8. Implementation of Sub-schema Homogeneous Structures.

While defining the schema facilities, 2 types of homogeneous structure were defined:-

(1) Structures definable by 1 or more secondary key attribute values e.g. employee within project structure.

(2) Structures definable only by explicit structuring from the user or owner of the data e.g. parts structure.

The first type is definable by specifying the value of the employee status. In the EMPLOYEE/PROJECT example described in the construct diagram (figure 2.17.) the PROJECT aggregate consists of a 'PROJECT MANAGER' and a 'TEAM' aggregate within it. In the schema a secondary key was defined to identify the manager by 'EMPLOYEE STATUS WITHIN PROJECT'. At the sub-schema level, the manager details are obtained by setting the project code as the key to the data search command (this will be described in the section on the data manipulation language) and by explicitly defining the 'EMPLOYEE STATUS' for the manager i.e. the definition of the level of employee for project management is explicitly defined in the sub-schema as follows:-

```
DATA AGGREGATE PROJ-MANAGER;
    ITEM EMPLOYEE-STATUS = EMPLOYEE.EMPLOYEE-STATUS etc.
        VALUE = 5 etc.
```

Such a value setting is constant i.e. any attempt to amend the value from a run-unit will raise an error.

The second type of homogeneous structure uses the idea of a secondary key which is defined not as a derivation of an attribute but as a STATUS-LEVEL key. This indicates to the B.B.M.S. that the programmer through his sub-schema must set the value of the key himself and the value will indicate the position of the data element within the structure. The secondary key is identified at the sub-schema by the following instruction:-

```
ITEM item-name = STATUS-LEVEL;
```

The item-name must be identical to the secondary key name defined in the schema.

The item-name must be defined in the access table (to be described in the next section) as a key.

The value that the key takes can be best illustrated by an example

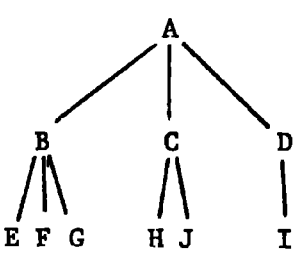


FIGURE 8.3.

If 'J' is required the value given to the key is:-

The numbers define the number of the branch down which access is to be made and the solidus indicates that the next levelshould be accessed.

No solidiimeans level '2' is the depth required, 1 solidus means level '3', '2' solidii mean level '4' and so on.

the first '2' points to 'C'
the second '2' points to 'J'

To access 'A' requires the key to be set to NULL.

8.5. THE PHYSICAL DATA DESCRIPTION LANGUAGE.

The description of data as physically stored need only define the characteristics of the physical grouping and linking as well as the storage characteristics of each data field. The definition of where data is allocated on the storage medium and what medium is used should be kept independent from the physical data definition.

Already it has been stated that data should be stored in subject files independent from how the data is used in the present or will be used in the future. To that intent, relationships, defined to access data to be used by many and varied applications, should be stored externally from the data records. In the BUSINESS example there are 3 subject files an'ORDERS FILE', a 'CUSTOMER FILE' and a 'PRODUCT FILE'. The method of definition of the physical characteristics of the data should be clear and simple in the same manner as the sub-schema and schema definitions.

8.5.1. The File.

The highest level of physical data grouping is the FILE, the format of the command is simply:-

FILE file-name;

8.5.2. The Record.

Within each file there may be several record types e.g. there could be several different types of order that can be placed with the organisation. The data content may differ so much from order type to order type that a different record type is required for orders of differing types. The format of the command is:-

RECORD record-name;

8.5.3. The Field.

The basic physical data unit is the field. It is the data that is derived from the schema. The method of derivation is that used in the sub-schema i.e.

FIELD field-name = attribute-name (a,b): attribute-name (x,y);

as with the sub-schema derivation several different attributes or parts of attributes, can be concatenated together to form the physical data unit. There is no need to specify relationship or entity identifiers because there may be logical data redundancy e.g. the order number is present in 2 entities but in such a case there is no need to have the order number stored twice and so at the instance of the schema all schema occurrence of a particular order number will map to this physically stored data field. The attribute may be qualified by an entity but this restricts mapping of the attribute in that particular entity to the physical data field. The attribute from more than one entity can be mapped to a single data field, the constituent entities in the multiple mapping being separated by commas. Any number of such entity/attributes can be mapped into a single data field. To illustrate what mapping definitions are possible

FIELD field-name$_1$ = entity-name$_1$.attribute-name$_1$ (a,b):
entity-name$_2$.attribute-name$_2$ (x,y),
entity-name$_3$.attribute-name$_1$ (a,b):
entity-name$_4$.attribute-name$_2$ (x,y);

In this example, 2 attributes with entity qualifiers are concatenated to form the stored data field. A second pair of concatenated attributes are similarly mapped to this stored data. A further stored field may be mapped as follows:-

FIELD field-name$_2$= attribute-name$_1$ (a,b):attribute-name$_2$ (x,y);

This statement indicates that all other schema definitions of these attributes in any other entities are mapped on to a second stored field.

Field Definition.

The maximum size of each field (the number of characters/digits) is implied by the mapping, however it must be possible to define how a field is stored. The definition of how a field is stored (character, unpacked decimal,

packed decimal, binary) is defined in the physical data description language. This field definition is, of course, machine dependent. It is defined by a MODE qualifier on the FIELD definition e.g.

FIELD field-name = attribute-name MODE UNPACKED

A check should be made that the storage mapping includes only characters that are defined as numeric at the schema level. The start position of each field relative to the start position of the stored record is implied by:-

(1) the order of field definition relative to other fields;

(2) the mode of the field e.g. if the field is binary, the field will be positioned on a word boundary.

8.5.4. Key.

Every record will have an identifier. This key will be mapped directly to a primary key in the schema e.g. ORDER NUMBER but the key may be a part of another or several other primary keys e.g. the ORDER NUMBER as part of PRODUCT NUMBER/ORDER NUMBER primary key of the 'product number/order number' entity.

The key identification at the stored record level is used to identify records for recovery and reorganisation.

8.5.5. Segments.

This is a grouping of fields within a record equivalent to the data aggregate and vector at sub-schema level. The purpose of the segment is twofold:-

(1) As a physical representation to define repeated occurrences (or possibly only one) of a set of fields. No specification of how many occurrences, from 1 to n, are expected, but the definition of a grouping within a stored record permits the definition of stored data hierarchies. Segments can also be defined within other segments, however overlapping of segments should not be allowed. An example of the need for a segment is the varying number of products that may be ordered within a single order.

(2) Records can grow to a significant size particularly if they consist of vector data as exists in the 'order' example. There is no reason why the size

of the record when stored should not exceed the size of the transfer unit, defined in Appendix 1 as the PAGE. This means that only part of a record can be transferred at a time. The segment definition is the break point or 'boundary marker' at which the data base access routines can split a record to enable part to be fitted into the available space in a page. The mechanics of this will be discussed later and obviously the best solution is to hold all data for one record in a single page.

The format of the segment instruction would be:-

SEGMENT segment-name;

and the grouping would be terminated by:-

SEGMENT END;

Like the full record, the segment can have a key to identify it. The record key is identified by being the only key that is not within a segment.

The form of the key qualifier is:-

KEY FIELD field-name = attribute-name etc.

8.5.6. Compaction.

One of the most important facilities to be provided with the description of data at the stored level is the provision of compaction techniques. The form of the instruction is

CODING procedure-name;

or CODING table-name;

The procedures can be user written for his own purposes but all the general procedures described in section 2.5.12. should be provided by the D.B.M.S. In addition to procedures, coding tables should also be provided, the most common example:-

```
TABLE SEX;
     1 = M;
     0 = F;
TABLE END;
```

This converts the single character real world representation to a 1-bit stored form.

The coding should be able to be applied at all levels of storage structure from the file to the field. The more generalised the technique the more suitable it becomes at the file level. If a coding procedure is introduced at a particular level all physical structures at a lower level would default to this compaction technique unless a different technique was explicitly called, for instance, for a particular type of field. If a particular physical structure at a lower level should require no compaction then a new argument to the CODING instruction is required.

```
CODING NONE;
```

Similarly a coding procedure could be introduced as a default or for a particular class of field within a specific file, e.g.:-

```
FILE file-name;
CODING BINARY procedure-name-1;
CODING UNPACKED procedure-name-2;
CODING procedure-name-3;
KEY FIELD field-name = etc.
```

The final CODING statement would define that all fields other than those in binary or unpacked format would be coded by procedure-name-3.

8.6. STORAGE CONTROL.

No description is included on where data is stored, the idea of an AREA of storage should not be part of the physical description of data because placement must be under the control of the D.B.M.S. itself.

It is not necessary to place the role of storage control fully on to the D.B.M.S. but facilities should be provided by the D.B.M.S. to report the accessing characteristics of data under its control to enable the data base administration staff to direct partial or full reorganisation whenever they wish. The data base staff must be able to define in a general sense the placement characteristics of data, for instance, records that are likely to be required should be clustered together on the central cylinders of the disc drives. The perfect example of this kind of data use, is in the airline reservation

systems when the records related to booking for flights in the immediate future are much more likely to require accessing than other later flight records. The date and time attribute values of each flight are significant in placement control.

In a different type of application, certain transactions may predominate during the normal data base accessing. During these transactions, certain record types from possibly different files will be accessed in a set sequence, in such cases the D.B.A. should be able to direct the storage of these records together, or certainly close enough to prevent head movement. The values of certain data fields probably the value of key fields will 'tie' such records together in this way. In the BUSINESS example, the CUSTOMER NUMBER field in the ORDER record will indicate that the CUSTOMER record with the same CUSTOMER NUMBER value should be stored close to the ORDER record.

The D.B.M.S. should allow records (and segments) from different files to 'intermix'. The physical data structure file should not imply that all records from the same file are stored together.

In conclusion on the topic of storage control, the D.B.A. should be provided with reorganisation procedures to produce the optimum data accessing solution. He should be able to obtain details of every record accessed and the sequence of access during the previous day's data base accessing. He should be provided, or should be able to easily generate his own statistical programs to clarify any aspect concerning the accessing and should be able by simulation techniques supplied by the D.B.M.S. to test out proposed alterations to the storage strategy. This facility would be most useful if new extensions to the data base accessing are planned (e.g. new transactions, new entities, new sub-schemas etc.)

8.7. DATA BASE MANIPULATION.

The essential aim of a good data base manipulation language should be to provide easy commands to access the data base but which are nevertheless powerful in their capability.

The commands should be instructions (verbs) able to be used in any existing or new application language. The commands are to provide the application programmer with exactly the data he requires based on the data types and structures defined in his sub-schema with the minimum of effort on his part.

The basis of this approach is to define a subset of the sub-schema associated with the program or subroutine with this subset defining the data that is required to be accessed. To illustrate this concept with an example, if the programmer wishes to print an order form for an order which has been placed with our example organisation, the program must read the whole order from the data base and possibly perform certain processing on the data before printing. The programmer will not know how many products are on order (the size of the order-product vector is not known) and to obtain all the data together would unnecessarily complicate the processing. The D.B.M.S. must provide the facility to read the order by using the order number value supplied by the programmer, to hold the data and to pass to the program's working data area only those parts of the order record that the program wishes to process. The manipulation language should provide the capability to access data directly or in this 2-stage manner.

8.7.1. Access Tables.

Access table is the name given to the subset of the sub-schema that requires accessing. The access table has the following characteristics:-

Content.

The data elements that are required to be accessed are listed in the table. If a data aggregate or vector is one of the data elements required, all data elements that are subordinate to the entry in the table are also included for access. If any subordinate data elements are to be excluded they must be named as follows:-

Sub-schema.	Access Table.	Data accessed.
AGGREGATE TEAM;	TEAM;	TEAM
ITEM EMPLOYEE-NUMBER ;	EXCLUDE EMPLOYEE-SALARY;	EMPLOYEE-NUMBER
ITEM EMPLOYEE-NAME;		EMPLOYEE-SALARY
ITEM EMPLOYEE-SALARY;		
AGGREGATE END;		

Access Identification.

Several methods of access should be provided, such as:-

Key - this is the simplest of methods of access identification. The key is identified in the following way:-

KEY data-element-name;

Selection - There may be the need to obtain more than one occurrence of a logical sub-schema record, often the choice of records is based on several criteria occurring. Selection itself may be only the first step, the user may wish records to be passed to him on the basis of different degrees of choice. An example of this type of selection might be the selection of 'orders' in the example, possibly to be output to a terminal printer as a picking list. The criteria for selection may be CUSTOMER PRIORITY linked in some way to the 'number of days the order is outstanding' provided all the products on order can be supplied. The procedure required would be something like:-

```
IF ORD-AVAIL = 2 (i.e. all products for this order are available)
IF DATE   -   ORDER-DATE > 3
     SELECT data-element-name
ELSE IF PRIORITY    >    6
     OR
ELSE IF DATE    -    ORDER-DATE > 2
     SELECT data-element-name etc.
```

The first selection scan is for orders more than 3 days old, the second scan is for priority orders irrespective of date, or an order more than 2 days old (i.e. 3 days old after first scan has been completed).

8.7.2. Data Base Commands.

The data base commands are designed to be very easy to use, the format is:-

data-base-command access-table-name.

The commands are as follows:-

FIND

This command obtains the data from the data base as directed by the data structure defined in the particular access table used and based on the sub-schema available to this routine/program.

The amount of data can be small or very large. The data obtained is not available to the user after execution of this command, the purpose of the command is to obtain all the data that is known to be required to be processed but possibly before it is needed for processing.

GIVE.

This command passes to the program part or all of the data obtained by the FIND or by several FIND's earlier in the program or possibly in an earlier program. The effect of the command is to set up the user areas within his program from the D.B.M.S. system area. The GIVE may come immediately after the FIND or considerably later in processing.

PUT.

This command is equivalent to the GIVE command with the difference that the data defined in the access table is going from the user program work area to the D.B.M.S. work area.

STORE.

This command is equivalent to the FIND. It does not necessarily mean that the data defined in the access table will be written to disc but it may only be written when the primary storage area the data is occupying is required by data from another FIND. Similarly the FIND command will look in primary storage first in case the data required is already in primary storage.

REMOVE.

This is the logical delete command. The purpose of the command is to delete the run-unit's knowledge of the particular data REMOVED. The data although available to other run-units is not available to the run-unit issuing the command from that point onwards.

DELETE.

This command physically removes the data defined in the access table making that data unavailable to all run-units. The action of both the REMOVE and the DELETE commands is on whole entity occurrences. Several entities of the same type or of different types may be deleted by one command.

SAVE AND RETRIEVE.

The effect of these 2 commands is equivalent to the write and read 'slot' defined in Appendix 2. The effect is to store working data that is defined within the access table (and consequently within the sub-schema) on to a temporary scratch file. It is in effect handling the overflow from the processing of a large amount of data, the data being required for further processing later or in the same program/routine. The difference between SAVE

and STORE is that the SAVE data is not processed by the sub-schema/schema and is not mapped back on to the data base. The slot commands detailed in Appendix 2 should still be useable but are available as non-data base data storage commands.

TRANSACTION.

This command is similar to SAVE in that the data is stored on a temporary file. However the purpose of the command is to cause the data base to be updated by the data defined in the access table at the time and date also specified in the access table. The date and time are defined as follows:-

DATE data-item-name;
TIME data-item-name;

The effect of the command is to provide a delayed updating facility to aid time consistency e.g. if a particular update must take place at a certain hour the D.B.M.S. will perform the update automatically without human intervention so that before that time the old values will be accessed and after, the new values would be available. In the BUSINESS example it may be that the customer address may be changed in this way, with the organisation receiving prior notification of the change and 'posting' it when the change was received so that the change would happen automatically on the date the customer stipulated.

REPORT.

This is a command very similar to the transaction in the way that it is an aid to time consistency. The purpose of the command is to 'lock-out' the parts of the data base necessary at the date and time specified in the access table and to select the data from the data base into a file from which the necessary time consistent reports could be produced. This is not the only method of producing reports consisting of time consistent data, the subject will be covered in further detail later in the chapter.

FORMAT.

A less rigorous form of the REPORT command. With this command the data defined in the access table is obtained immediately and the data is stored in a temporary file. The data obtained can then be printed, totals produced and the report layout set up by using a report program generator which is a utility program provided with the D.B.M.S.

The report program generator as well as providing report layout facilities should also provide standard functions and the capacity to permit the user to supply his own set of function procedures. Examples of functions are COUNT, MAX, MIN, AVERAGE, MEAN, STANDARD DEVIATION etc.

8.7.3. Null, Default and Initial Values.

Null.

The setting of a data element to a NULL value indicates that the particular data element does not exist. It provides a simple method of removing a data element from the data base.

Two subroutines are used to:-

(a) set a NULL value:-

CALL SET-NULL data-element-name.

(b) check for a NULL value:-

CALL CHECK-NULL data-element-name.

Default.

Any data item in the sub-schema may be given a default value. The default value may be set up in the data item by the following application language statement (COBOL).

CALL DEFAULT data-item-name.

The default value may be changed easily without effecting a recompilation of the program. The default can be used for attributes at the schema level but any conflicting sub-schema default value will replace the schema value for any program using such a sub-schema.

Initial.

Any data element may be assigned an initial value within the sub-schema. The effect for those data elements with initial values is that at the start of a run-unit and after a PUT command (copy of data from a run-unit working area to the D.B.M.S. buffers) the data elements having INITIAL values will be set to those values. If such values are not over-written by reading data (GIVE command) or from the effects of application program processing then when the data is PUT the initial values not changed will be written to the data base. The effect is that for any data elements with initial values, data set up before a PUT command cannot be assumed to remain unaltered after the command when they will revert to the initial value.

The format of the instruction is:-

INITIAL = explicit value;

similarly for a default value:-

DEFAULT = explicit value;

8.7.4. Accessing Homogeneous Structures.

In the section describing the sub-schema, the setting up of the STATUS LEVEL was described. This data element is described as a key data item in the access table as follows:-

STRUCTURE KEY data-item-name;

The facility is also provided to access in a single request part or whole of the homogeneous structure by the command:-

STRUCTURE FROM KEY data-item-name;

The data may be held in the D.B.M.S. main store buffers or in the temporary file.

An extra facility is provided for such complex accessing in the way that the data may be passed to the user work area. For this or any other complex structure obtained by a single FIND request the following qualifiers can be added to any GIVE, PUT, REMOVE or DELETE command (i.e. any command transferring data to/from the user work area):-

NEXT, PRIOR, FIRST, LAST or numeric literal.

thus:-

GIVE NEXT access-table-name;

NEXT tells the D.B.M.S. that the next data identified by the key is to be passed to the program. If GIVE FIRST was used after the 5th. logical set of data had been obtained, the D.B.M.S. would go back and reset its pointers at the first set. The NUMERIC LITERAL represents the number of the set entry next required.

To enhance data independence, the qualifier may be situated within the access table if required:-

NEXT KEY data-element-name;

8.8. EXAMPLES OF THE FACILITIES AT WORK.

Two particular examples that illustrate a certain number of proposed facilities are:-

(1) adding new orders;

(2) producing a picking list of products for orders.

8.8.1. Adding New Orders.

The sub-schema can be depicted in construct diagram form as follows:-

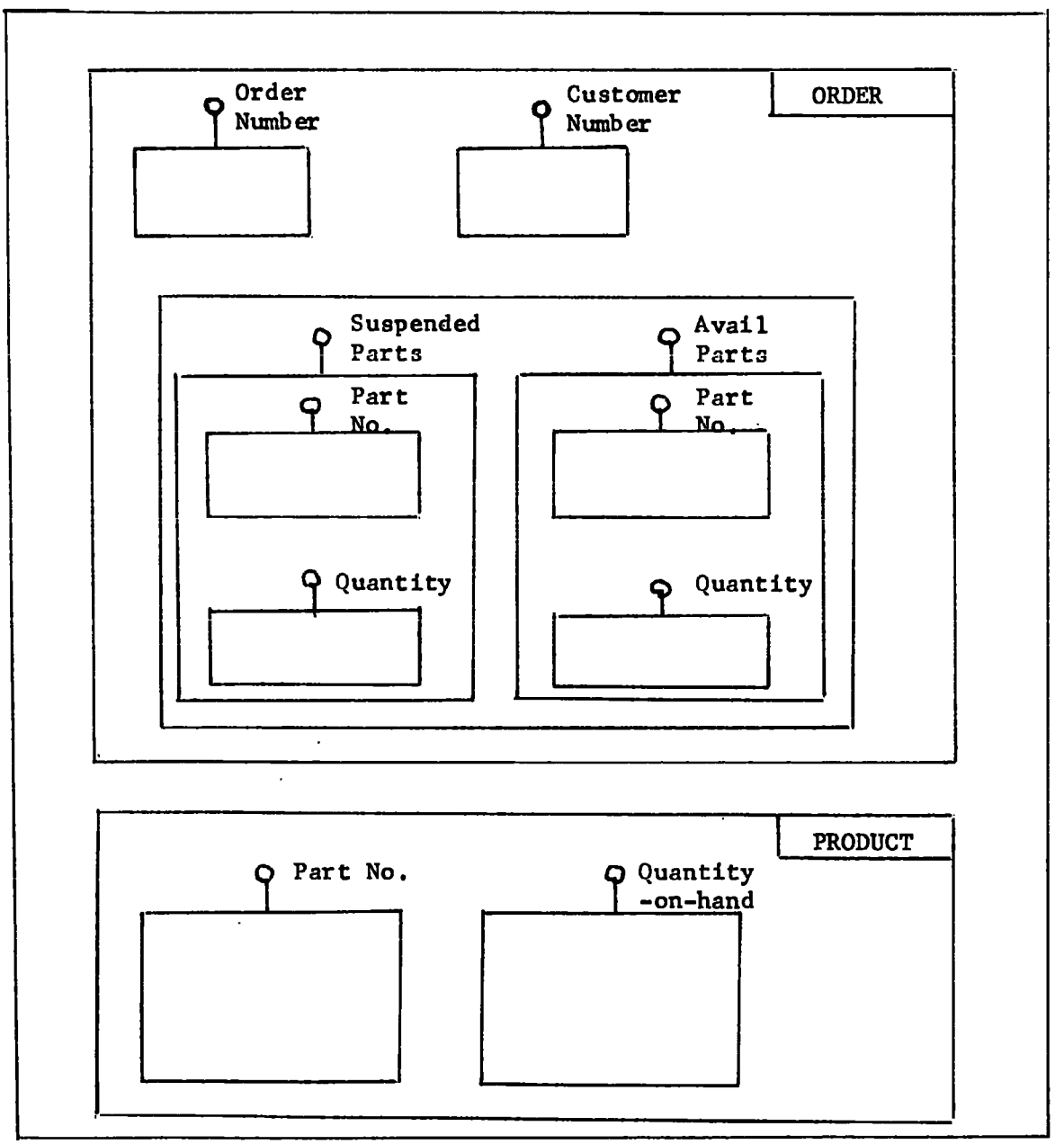


FIGURE 8.4. CONSTRUCT OF A NEW ORDER SUB-SCHEMA.

Describing the sub-schema entries briefly:-

```
SUB-SCHEMA NEW-ORDER;
   RECORD ORDER-DETAILS;
   ITEM ORDER-NUMBER = ORDER. ORDER-NO etc.
   ITEM CUSTOMER-NUMBER = ORDER. CUST-NO etc.
   VECTOR ORDER-PARTS;
   AGGREGATE SUSPEND-PARTS;
     ITEM PART-NO = SUSPEND? ORDER-PROD. PART-NO etc.
     ITEM QUANTITY = SUSPEND? ORDER-PROD. QUANTITY etc.
   AGGREGATE END;
   AGGREGATE AVAILABLE-PARTS;
     ITEM PART-NO = AVAIL? ORDER-PROD. PART-NO etc.
     ITEM QUANTITY = AVAIL? ORDER-PROD. QUANTITY etc.
   AGGREGATE END;
VECTOR END;
RECORD PRODUCT-DETAILS;
     ITEM PART-NO = STOCK? PRODUCT. PART-NO etc.
     ITEM QUANTITY-ON-HAND = STOCK? PRODUCT. QOH etc.
SUB-SCHEMA END;
```

The run-unit which sets up the new order has input to it ORDER-NUMBER, CUSTOMER-NUMBER and several PART-NO's and QUANTITY's.

Some of run-unit's processing instructions are as follows:-

```
MOVE ORDER-NUMBER OF INPUT TO ORDER-NUMBER OF ORDER-DETAILS.
MOVE CUSTOMER-NUMBER OF INPUT TO CUSTOMER-NUMBER OF ORDER-DETAILS.
PUT ORDER-HEADER.
```

where

```
ACCESS-TABLE ORDER-HEADER;
 ORDER-NUMBER;
 CUSTOMER-NUMBER;
 KEY ORDER-NUMBER;
 TABLE END;
```

The ORDER HEADER details are first set up and transferred from the run-unit's working storage area to the D.B.M.S. system buffer using the ORDER NUMBER as key.

```
MOVE PART-NO OF INPUT TO PART-NO OF PRODUCT-DETAILS.
FIND PRODUCT.
GIVE PRODUCT.
```

where ACCESS-TABLE PRODUCT;
PRODUCT-DETAILS;
KEY PART-NO;
TABLE END;

Each PART NUMBER is taken in turn and the 'QUANTITY-ON-HAND' is read into the run-unit's working storage using the PART NUMBER as key.

```
IF ERROR-REPLY NOT = ZERO GO TO ERROR-ROUTINE.
CALL CHECK-NULL PRODUCT-DETAILS
IF NULL-REPLY NOT = ZERO (i.e. NULL is set)
MOVE PART-NO OF INPUT TO PART-NO OF SUSPEND-PARTS
MOVE QUANTITY OF INPUT TO QUANTITY OF SUSPEND-PARTS
PUT SUSPENSION.
IF NULL-REPLY = ZERO
MOVE PART-NO OF INPUT TO PART-NO OF AVAILABLE-PARTS
MOVE QUANTITY OF INPUT TO QUANTITY OF AVAILABLE-PARTS
SUBTRACT QUANTITY OF INPUT FROM QUANTITY-ON-HAND OF PRODUCT-DETAILS
PUT PRODUCT
STORE PRODUCT
PUT AVAILABLE.
```

```
ACCESS-TABLE SUSPENSION;
SUSPEND-PARTS;
KEY PART-NO;
TABLE END;
```

```
ACCESS-TABLE AVAILABLE;
AVAILABLE-PARTS;
KEY PART-NO;
TABLE END;
```

The error reply is set if the PART NUMBER does not exist. The check for the setting of the NULL switch is because if it is set the PRODUCT with the PART NUMBER requested does not belong to the STOCK relationship i.e. the product belongs to the 'BUY-IN' relationship consequently itis not available for ordering, the ORDER for that PRODUCT must be put into the SUSPEND relationship. The ORDER-PRODUCT details are transferred from the program's working storage to the system buffers of the D.B.M.S. using the PUT command. If the PRODUCT is in STOCK the ORDER-PRODUCT is made AVAILABLE. The QUANTITY of the PRODUCT ordered is subtract from the QUANTITY-ON-HAND for that PRODUCT before the PRODUCT details are written back to the data base using the PUT and STORE commands and with a reduced QUANTITY-ON-HAND value.

When all the PRODUCTS that are on order have been processed the ORDER is actually written to the data base with the store command:-

```
            STORE NEW-ORDER.
where  ACCESS-TABLE NEW-ORDER;
         ORDER-DETAILS;
       TABLE-END;
```

All the ORDER-DETAILS are written at this point. There is no need for the definition of keys because they have already been defined in the individual PUT commands.

When the STORE command is executed the sub-schema and schema integrity and access control checks are performed on the data before finally the 'order' record is written to the 'orders' file.

The checks that would be performed are:-

- to check that the user and/or terminal has the capability to write this record type;
- to check that for integrity:-
  the CUSTOMER NUMBER is in the data base;
  the PART NUMBER is in the data base;
  the ORDER NUMBER is unique;

all these checks have already been described. They are defined at the sub-schema level.

- to check that the QUANTITY-ON-HAND of each product is not less than its corresponding QUANTITY-LEVEL.

The procedure to be called is as follows:-

```
PROCEDURE CHECK-LEVEL;
   IF QOH < QUANTITY-LEVEL
      RELATIONSHIP SYSTEM = BUY-IN
   ELSE
      RELATIONSHIP SYSTEM = AVAIL
   PROC END;
```

8.8.2. Picking List Processing.

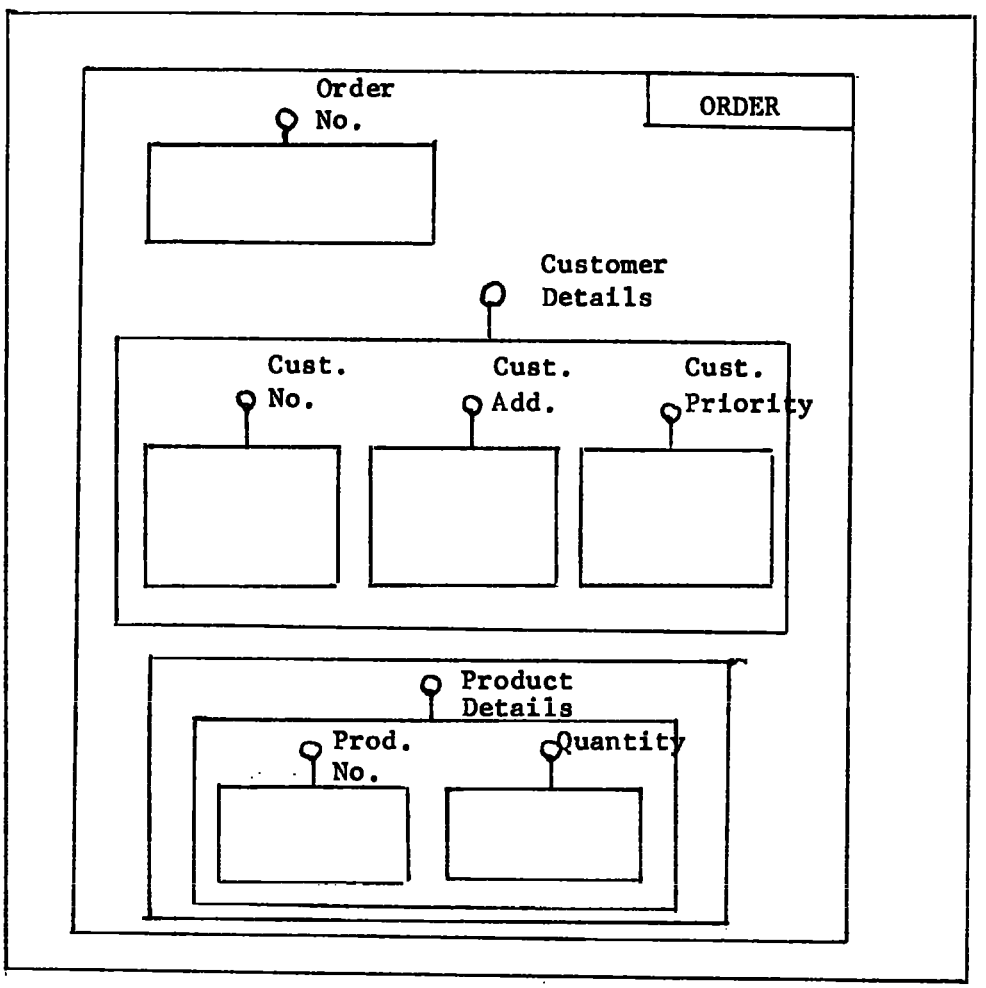


FIGURE 8.5. CONSTRUCT FOR PICKING LIST.

The ORDER NUMBER and CUSTOMER DETAILS are collected into a temporary file on the basis of the selection procedure detailed in section 8.7.1. The FIND command would be used. The sub-schema would have defined within it a logical file which would be equivalent to the file used in the SAVE command, in effect, the file acts as a much larger D.B.M.S. buffer store.

Once selected each order would be brought into the run-unit's working storage by a GIVE command. The picking list would probably be produced on a terminal printer in the warehouse. The ORDER NUMBER would be printed and all the products would be read from the data base for that order by using a FIND command. Each product would be passed to the run-unit and output to the printer individually. Facilities for outputting to the printer would be provided by the T.P. monitor used in conjunction with the D.B.M.S.

The picking list would indicate which products were to be picked and in which order, it would not be implied by being output on the list that all the products would be picked. Only enough products may be picked to fill the lorry that will make the deliveries. As a complete order is loaded on to the transport, the confirmation of the picking would be made at the warehouse terminal and the invoicing system would use the data base data to print an order invoice on possibly a different printer which has pre-printed invoicing stationery.

The picking list processing is an example of on-line processing that takes place in background. The initial request is made for a picking list possibly based on an area code which indicates where the particular lorry is delivering. A confirmatory response should be made within seconds but the production of the list may take several minutes as background processing to other transaction processing.

8.9. THE ENVIRONMENT TO SUPPORT THESE D.B.M.S. FACILITIES.

The function of the data base administration team is said to be 60% non-technical mainly involved with data analysis and 40% technical involving D.B.M.S. performance, security, recovery and general implementation. This section deals with the non-technical aspects involved with implementing a D.B.M.S. based on the structure and facilities described already in this chapter.

8.9.1. Data Analysis.

The administration team should consist of several data analysts, probably ex-systems analysts, who will be responsible for the design of the schema structure. Different data analysts will be working in different development areas, however with the likely integration of several developments based on a single schema, the

data analysts must be centralised working together with data analysts from other branches of development to ensure an efficient data base design. They will be 'loaned out' to development teams to act in a consultative capacity, directing how the development of a system should progress to fit and not to be in conflict with the data base design philosophy. They will be responsible for information flow analysis and defining the entities and attributes involved, fulfilling a similar function as the senior systems analyst in the organisation starting out on data base implementation described in chapter 7. It is a partnership between the systems analyst and data analyst, with the systems analyst in overall charge of implementation. There may be differences of opinion, with the data analyst not agreeing with a system design point which might be in conflict with the data base design philosophy. Such problems would be resolved at a higher level.

The main responsibilities of a data analyst are:-

(1) To define the information flow involved in the system being developed.

(2) To specify the likely volume of information.

(3) To define the entities and attributes required in the schema and the possible changes to the current schema.

(4) To resolve any political problems concerning the access of data and to advise both user and systems analyst.

The systems analyst and data analyst will produce an initial systems design document which will include the outline systems design as well as the above data analysis information.

The two analyst should be at project leader level.

8.9.2. Data Base Design.

The data analyst who did the initial study will probably supervise the work of 1 or more data base designers who will do the more detailed data base design, working with and advising their counterparts in the system development team. They will define all the characteristics of the schema, the integrity, data validation and access control requirements. They will work with the system designers at the program design level to specify the sub-schema's and access tables required. The definition of the physical data description will also be their responsibility. This ends another phase of development with the detailed systems design document including both system and data base design.

8.9.3. Data Base Programming.

A commonly held view is that the data base administrator designs and maintains the schema while the analysts and programmers look after the sub-schema. This conceptual D.B.M.S. attempts to put more emphasis on a combined approach between systems and data administration staff. This is down to the level of programming. The data base programmer will code the schema, sub-schema and physical data language statements.

8.9.4. The Use of the Data Dictionary.

The data dictionary has been defined by many people as the data base of data bases. The D.B.M.S. supplier should provide easy to use facilities (probably terminal conversations) to enquire on any information held in the data dictionary.

There need be only one compiler for all 3 data description languages. The essential structure of the languages is the same. As a by-product of the compilation process both the data dictionary and the directory will be updated. The dictionary will be no different to any other data base. The schema for the dictionary data base will be set up by the supplier and the details of schema and sub-schema instructions which define the dictionary data structures will be provided by the supplier as part of the documentation. The schema will be able to be altered by the user to change the way the data dictionary is structured. The user should also be able to add new sub-schema's of his own, to enable additions to be made to the dictionary enquiry system.

8.9.5. Testing Facilities.

Additions to any of the data description languages may be made to permit the testing of small alterations in data structure. Any data structure from an attribute upwards may be set up in one of two extra modes:-

TESTING - this mode indicates that the particular data described in this way will not be used to update or write to the live data base but will only simulate modification.

PARALLEL - this is a similar testing mode except that the data access for the data described in this way will be performed twice. The new version described in the PARALLEL mode will only simulate alterations to the data base, however it would provide a comparison feature enabling the live and testing versions to be compared.

8.9.6. The Affects on Department Structuring.

There is a trend for data processing departments to change their name to management services. The name more accurately represents the function the department performs. No longer is data simply processed, it is also administered.

The different section within this department each with their own managers reporting to the Management Services Manager should be:-

- Operations and Method Study;
- Systems Development;
- Data Administration;
- Computer Operations;
- Technical Services.

Operations and Methods would continue their current function of investigating current methods of working and possibly suggest organisational or system changes possibly with the development of a manual or a computerised system.

Systems Development Section is responsible for the implementation of computer systems. The section would consist of system analysts and programmers. The levels of staff under the Development Manager would be:-

- Project Leaders;
- Systems Designers;
- Programmers.

Data Administration would have 2 main functions:-

(1) Data analysis and data base design.

(2) Technical data base implementation and support.

The first function has already been covered in some details, the different staffing levels would be:-

- A Chief Data Analyst to supervise the function;
- Data Analysts;
- Data Base Designers;
- Data Base Programmers.

The second function would deal with the technical aspects of data base implementation. This function will be dealt with in more detail in the next chapter.

The Computer Operations Section would include:-

- computer operations itself;
- operating system and other system software support;
- programming maintenance.

The 'other system software' may include control over the introduction of T.P. monitor or D.B.M.S. releases, as well as assistance with performance measurement and tuning. This would inevitably involve close liaison with data base administration staff.

Programming maintenance will involve maintenance of systems other than extensive system rewrites. To be a maintenance programmer should be a higher position than data base programmer which itself should be a higher position than development programmer. A maintenance programmer should have worked in both the other two programming positions to enable him to have a wide knowledge. In a real sense the responsibilities of the position are far greater than in the development and data base programming positions.

Technical Services should provide advice on the technical aspects for the organisation both for the present and future. They must provide advice on the impact of developing technologies with an up-to-date awareness of the development in information technology. The staff will probably specialise in a particular area and work alone, although occasionally they may work together, for example, on a hardware evaluation project. They must be careful not to 'stand on the toes' of technical staff in the operations and Data Base Administration Sections. They will probably have already worked in one of these two sections.

9. IMPLEMENTATION OF THE FACILITIES.

The essential objectives in implementing any D.B.M.S. are:-

(1) To maintain a secure data base.

(2) To try to provide the best possible flexibility without affecting performance and to try to provide performance levels as good as the performance level achievable without a D.B.M.S.

The ideas expressed in this chapter are wherever possible machine independent. Any complex piece of software should be designed to a target design which is machine independent but which can be fitted to any machine in the easiest manner possible.

The basic implementation philosophy is to provide as much flexibility as possible based on a data base dictionary. The control of data base management mechanics and data base access is by 'directory records' which are derived from the dictionary contents.

9.1. THE DATA BASE DICTIONARY.

9.1.1. Dictionary Entries for the Schema.

The following tables are involved in describing the schema:-

- The attribute table.
- The entity table.
- The entity/attribute table.
- The schema table.

The Attribute Table.

This table contains an entry for every attribute in the schema. The entry describes the characteristics of the attribute. Whether or not the attribute appears in more than one entity is irrelevant, there can only be one description for the attribute.

Each entry consists of the following elements:-

(1) An entry word count.

(2) The attribute identifier by which the attribute will be known to the D.B.M.S.

(3) The attribute name.

(4) The number of characters in the attribute either as a fixed value or as a maximum. This value has no correlation with the length as expressed either in storage or in the application program.

(5) The definition of class for each of the characters. There are 3 possible classes:-

- numeric
- alphabetic
- alphanumeric.

Only 2 bits are required to define the class of each character. In addition 2 bits are required to define whether the attribute may have:-

- negative only values;
- positive only values;
- either positive or negative values are permissible.

(6) (a) The remaining element consists of a header, consisting of:-
- an element code (i.e. indicates whether validation, coding etc.);
- the element length.

The first element is the 'validation' element consisting of:-

(b) The type of validation in use i.e.:-
- range
- value
- list
- procedure

(c) Range - This element consists of the 2 range values. Each range value will correspond to the length specified for the attribute.

(d) Value - There is a maximum of 5 values possible. The element of the entry will consist of:-

- the number of values present;
- the values, the value length will correspond to the length specified for the attribute.

(e) List - The element consists of the following structure:-

- an indicator to identify which, if any, set operations are to be performed on the lists i.e. RESTRICTION, COMPOSITION, JOIN, SEQUENCE or none;
- the number of lists involved;
- the list identifier (the structure of the list will be described later in the chapter).

(f) Procedure - Details of describing procedures and their calls will be described later.

(g) An indicator to identify whether the result of the validation rules provide a set of valid or invalid values.

(h) The structure of the remainder of the 'validation' element is dependent on the validation method.

(7) (a) The second element is the 'coding' element. The element has a header as described in paragraph (6a).

(b) The element has an indicator which defines by which method the coding is performed:-

- by a table of values;
- by procedure.

(c) The remainder of the entry is dependent on whether a table or a procedure is used.

(8) The third element is the 'virtual' element which, if used, defines that the value of the attribute is derived not from a stored field but by derivation from attribute values.

The constituent parts of the element are:-

(a) The element header (paragraph 6a).

(b) The remainder of the entry is in the format for a procedure.

(9) The final element describes a default value should one exist. It consists of:-

(a) The element header (paragraph 6a).

(b) The default value.

The Entity Table.

The entity table has an entry for each entity within a particular schema. The possible elements of each entry are as follows:-

(1) The entity entry word count.

(2) The entity identifier by which the entity will be known by the D.B.M.S.

(3) The entity name.

(4) The number of attributes in the entity.

(5) An element for each attribute in this entity, defining:-

(a) The attribute identifier.

(b) The attribute's start position defined as the number of characters from the start of the entity.

(6) The definition of the primary key. The entry is structured as follows:-

(a) The primary key code which indicates this element of the entry is a primary key.

(b) The element length.

(c) The primary key identifier which the D.B.M.S. uses as the key identifier rather than the key name.

(d) The primary key name.

(e) The number of attributes that make up the primary key.
For each attribute which makes up the key:-

(f) The attribute identifier.

(g) The start character and the number of characters within the attribute that are included as part of the key; if the whole attribute is used in the key, these values will be zero.

(7) The definition of the secondary keys. The structure of these elements are as follows:-

(a) The secondary key code which labels this element as a secondary key.

(b) The element length.

(c) The secondary key identifier.

(d) The secondary key name.

(e) The primary key can only be derived by attribute mapping (as described in paragraphs (6f) and (6g)), however, the number of different methods of deriving a secondary key are far greater, and an indicator is required to identify which method is used. The possible methods are:-

- by procedure;
- by map;
- using the 'WITHIN' clause for homogeneous structures;
- using 'STATUS-LEVEL' for homogeneous structures.

(f) The map entry is the same method as for the primary key (paragraphs (6f) and (6g)).

(g) The procedure entry is described later in the chapter.

(h) The 'WITHIN' entry consists of

- the number of secondary keys involved in the WITHIN clause (the depth);

- the secondary key identifiers which are used as components; they are in the same sequence as they are described in the 'WITHIN' statement in the above schema.

(i) No further entry definition is required for 'STATUS-LEVEL' secondary keys.

(j) More especially for the mapping and procedure methods, there is the possibility of a 'LIST' of value ranges applying to a single secondary index rather than having an index for one key value. An indicator is required to define whether such a list is being used. If a list is used items (k) and (l) are present.

(k) The list can refer to 1 of 3 types of values
- alphabetic;
- alphanumeric;
- numeric.

(l) The method of defining the list is described later in this chapter.

(8) If an integrity call is required the following part of the entity entry is included:-

(a) The code required to indicate an integrity is requested (e.g. INTE).

(b) The procedure call description (described later).

(9) Similar to (8), to indicate the need for a privacy call:-

(a) The privacy identity code (e.g. PRIV).

(b) The procedure call description (described later).

The Entity/Attribute Table.

Because the same attribute can belong to different entities there is the need to be able to quickly refer to which entities are described by an attribute.

Each entry to this table consists of an identifier which is a concatenation of the attribute identifier followed by the entity identifier. The entries are in identifier order so that all references to an attribute are grouped together.

The structure of each entry is as follows:-

(1) The entry identifier described above.

(2) The start character position of the attribute within the entity.

(3) Details concerning the directory records which are affected by this attribute. Directory records will be described in more detail later, but this entry consists of:-

(a) The number of directory records associated.

(b) The identifier of each record.

The Schema Table.

This is the highest level table in the schema dictionary. Each entry describes each schema known to the 'SYSTEM'.

The contents of each entry are as follows:-

(1) A schema identifier by which the schema is known internally to the D.B.M.S.

(2) The schema name.

(3) The location (address) of the entity table for this schema.

(4) The address of the attribute table for this schema.

(5) The address of the attribute/entity table for this schema.

(6) The description of relationships between entities. This description is as follows:-

(a) The number of relationships in the schema, followed for each relationship by:-

(b) The relationship name and identifier.

(c) The relationship type, being one of the following:-
- 1-to-1 relationship;
- 1-to-n relationship;
- m-to-n relationship.

(d) The 2 entity identifiers which are related, the 'owning entity' is the first of the 2 identifiers.

(7) The 'Entity/Field Table' address (this will be described later).

9.1.2. 'LIST' Description.

Each list contains values or value ranges. The list can be called at any appropriate point in the schema or sub-schema description. The structure of the list is as follows:-

(1) The number of entries. The two values that describe'the range'are considered one entry.

(2) A bit map which has a bit value for each entry:-
- '0' indicates the entry is a value;
- '1' indicates the entry is a range.

(3) For each'value'entry, the following structure is used:-
- length of entry;
- value.

(4) For each range entry, the following structure is used:-

- length of entry;
- the 2 values that provide the range limits.

9.1.3. 'PROCEDURE' Description.

Procedure Call.

Each procedure call requires describing in the following manner:-

(1) A procedure identifier which identifies the procedure internally to the D.B.M.S.

(2) The number of parameters in the call.

(3) If there are any parameters a bit map with a bit for every parameter. The 2-state values are:-

- '0' - the parameter is a literal;
- '1' - the parameter is a data name identifier (i.e. a data element or an attribute dependent at what level of data description the procedure is called).

For each parameter:-

(4) The parameter length.

(5) The parameter value (literal value or identifier value).

Procedure Description.

The procedure consists of a header followed by the procedure body.

(1) The procedure identifier which is used internally to identify which procedure is being called.

(2) The procedure name.

(3) The number of parameters.

(4) The address of the procedure in the routine library.

(5) The body of the procedure which is converted to machine instructions in a 2-stage process:-

(a) The first stage converts the procedure instructions into a summarised (compact) form which is a machine independent target code. This version is held in the dictionary.

(b) A compiled version in the machine language stored as a library routine.

The first version is easily convertible back to the original form should it be requested through the dictionary system for documentation purposes.

Using the 'date' procedure example in section 8.3.3, the following is the same procedure as it may appear in coded form:-

```
Ta␣tab-no;
If␣%1␣>␣TA␣tab-no(%2)
   FA␣failure-number
ELIF␣%3␣N=␣2␣EN
ELIF␣%1␣N=␣29␣EN
ELIF␣%3␣=␣00␣EN
EL␣W1␣=␣%3/4
IF␣RE␣=␣0␣EN
   FA␣failure-number
PE
```

In the original example 'INVALID' was used, this is not sufficient, a fail number must be set when a failure occurs to identify to the D.B.M.S. and possibly to the run-unit (dependent on the error level) the reason for the failure. 'W1' is identified as a working area local to the procedure. 'RE' is the code to represent the remainder value.

9.1.4. 'TABLE' Description.

The 'TABLE' is almost identical in description to the 'LIST', the difference being that there is more than one value for each entry. There must therefore be an extra description in the 'TABLE' entry to define how many values there are per entry.

9.1.5. Dictionary Entries for the Sub-schema.

The Item Table.

This table contains an entry for each data item in the sub-schema. The structure of each entry is as follows:-

(1) The item identifier by which the data item is known to the D.B.M.S.

(2) The item name.

(3) An indicator which identifies whether:-

- the item is derived directly from the schema, or,
- the item is subordinate to an aggregate which is itself mapped to the schema.

(4) The number of attributes in the mapping.

(5) The mapping is described as follows:-

- the one or more relationship identifiers (if any are used);
- the entity identifier;
- the attribute identifier;
- the attribute qualifier.

Because of the different mapping structures possible, each of the 4 types of mapping qualifier must be individually identifiable i.e. the first 2 bits of the identifier are used as follows:-

- '10' indicates a relationship identifier;
- '11' indicates the entity identifier;
- '01' indicates the attribute part qualifier;
- '00' indicates the attribute identifier.

The attribute identifier always appears last in the mapping stream because there will always be an attribute in the mapping and so this is a means of identifying the end of the identifiers.

(6) If the item is subordinate to a mapping element the only requirement is to specify the length of the item in characters.

The remaining 4 parts of the entity structure identify possible values associated with the item.

(7) If there is an 'INITIAL' value, the structure is:-

(a) An identifying code-'INIT';

(b) The value.

(8) If there is a 'DEFAULT' value, the structure is:-

(a) an identifying code-'DEFA';

(b) the value.

(9) If there is a fixed 'mapping value' associated with the element i.e. to identify a homogeneous structure e.g. the 'project manager employee'

(a) an identifying code-'VALU';

(b) the value.

(10) If there is a homogeneous structure of the 'STATUS-LEVEL' type there is simply an identifying code - 'STAT'.

The Structure Table.

The structure table defines the sub-schema's structure.

Data elements other than the data items are named at this level, together with the structure levels formed by these data elements and the data items described in the Item Table.

The contents of the table are as follows:-

(1) The entry is identified by one of the following codes:-
- FIL␣ (file)
- REC␣ (record)
- AGG␣ (aggregate)
- VEC␣ (vector).

(2) The identifier of the data element.

(3) The data element name.

(4) An indicator which is set when the data element is a direct map from the schema rather than an amalgamation of lower level data items that have been mapped.

(5) The mapping details should they exist. They will be structured in the same manner as for data items.

(6) Any data items directly below a particular data element entry must be identified by the particular data item identifiers.

(7) Any data element or data item can have associated with it (association is by position) integrity or access control checking. For integrity the structure of the command:-

- identifying code - 'INTE';
- the procedure call (as described in section 9.1.3).

(8) Access control has a greater number of options, the access mechanism is structured as follows:-

(a) The identifying code - 'PRIV'.

(b) An indicator to identify the chosen access mechanism, these are:-
- procedure;
- list;
- level;
- table.

(c) The particular details of the list, table, or procedure call as described in earlier sections. The 'level' has a value in the range 0-15 to indicate the particular access level of the data element.

(d) An indicator to identify whether the result of the access mechanism is 'INCLUDED' or 'EXCLUDED'.

(e) An indicator to identify whether 'OUTPUT' is permitted.

(f) An indicator to identify what access rights are, or are not permitted, these are:-

- read;
- write;
- logical delete;
- physical delete.

(g) Whether access is allowed (or restricted) by terminal or by user.

(9) Each structure has its own 'end code' which signals the end of a structure they are:-

- FILE (to terminate a file structure);
- RECE (to terminate a record structure);
- AGGE (to terminate an aggregate);
- VECE (to terminate a vector).

The Sub-schema Table.

There is an entry in this table for each sub-schema. The entry consists of:-

(1) The sub-schema identifier.

(2) The sub-schema name.

(3) The identifier of the schema which this sub-schema is using.

(4) The address of this sub-schema's item table.

(5) The address of this sub-schema's structure table.

9.1.6. Physical Data Entries in the Dictionary.

The Field Table.

The table consists of an entry for every field in a particular file.

The contents of each entry are as follows:-

(1) The field identifier.

(2) The field name.

(3) The mapping structure, defining which entities and attributes map to this field. Only the attribute identifier is mandatory. The same mapping method is used at the physical level as at the sub-schema level with the exception that relationships are not included at this level.

(4) A MODE indicator which identifies what type of storage mode is used for the field (e.g. binary, character, packed decimal etc.).

The File Structure Table.

This table is comparable to the structure table at the sub-schema level. It describes each record/segment structure within the file.

The contents of the table are as follows:-

(1) An indicator to identify the particular entry. The possible alternatives are:-

(a) Record - identified by 'REC␣' and its extent is terminated by a 'RECE' entry.

(b) Segment - identified by 'SEG␣' and its extent is terminated by a 'SEGE' entry.

(c) Key - identified by 'KEY␣'. This entry has no extent.

(2) The description of a record entry is:-

(a) Record Identifier.

(b) Record Name.

(3) Similarly, a segment and key entry both have an identifier and a name.

(4) The key entry also has a description of which fields defined in the 'Field Table' are key fields. The description consists of

(a) The number of fields in the key.

(b) The field identifiers that make up the key.

(5) Each non-key field is situated in its correct relative position within a segment or record. The field is simply represented by its identifier.

(6) At any point in the structure a compaction code can be applied to a field, key, segment or record. There are 3 possible coding descriptors, as follows:-

(a) CODP - coding is by procedure.

(b) CODT - coding is by table.

(c) CODM - coding is by procedure for a particular mode of field. This descriptor also identifies the 'mode type'.

Each descriptor is followed by either the table details or procedure call details.

The File Table.

Each entry describes a file as follows:-

(1) The file identifier.

(2) The file name.

(3) The 'Field Table' address.

(4) The 'File Structure Table' address.

The Entity/Field Table.

This table has a structured set of entries giving the entity identifiers of all entities in a schema. For each entity all the attribute identifiers are listed immediately after the entity to which they belong. For each attribute, all the field identifiers are included which map to this attribute.

The purpose of this table is to assist 'binding', which will be described in more detail later.

9.1.7. Access Table Entries in the Dictionary.

Each table has a header followed by the accessing information. The header consists of:-

(1) The table identifier.

(2) The table name.

(3) The identifier of the sub-schema to which this access table has access.

There is an entry for each data element in the access table with details as follows:-

(4) The data element number referred to in the table.

(5) An indicator determining whether this data element is to be included or excluded from the access.

The trailer entry gives details of the access key as follows:-

(6) KEY␣ or SEL␣ indicate that this is the KEY entry or SELECT entry respectively.

(7) For key access, the data element number is stored and for the selection access, the procedure number and details of procedure parameters are included.

9.1.8. Dictionary Set Up Facilities.

The dictionary entries that have been described are automatically produced as a by-product of interpretation of DDL commands. The D.B.A. team, as the user of the DDL, is able to change any part of any of the data descriptions by including only a subset of the description. As an example, to change characteristics of an attribute, requires all attribute details of only that attribute to be input. The interpreter identifies where the change is to take place by searching for an existing attribute with the same name. To change the attribute's name requires some extra identifier (e.g. CHANGE-NAME). There is a requirement also to delete an attribute. Respecification of the whole entity is required for:-

- inserting a new attribute;
- changing an attribute position within an entity;
- changing key information.

A short form of all the other attributes in the entity is required for inserting a new attribute or changing an attribute's position. The short form is 'ATT attribute-name' without any further details and no 'ATT END'.

The input of only part of a data description language is not for the benefit of the D.B.A. who will have editting facilities but to save processing time in setting up all the dictionary entries for a small change to an attribute.

Output from the dictionary system will be a full schema/sub-schema/file listing. With the use of the different tables it is a simple task for the software to obtain the source from the internally stored form.

On the same principal any enquiry may be made on any part of any data description held in the dictionary by using a terminal query processor to interface to the dictionary.

9.1.9. Other entries to the Dictionary.

The D.B.A. may input further dictionary details to describe the user programs and routines in existence. He is able to define which programs use which sub-schemas and access tables. This can be used as a check at compilation time. In this way the D.B.A. can examine if a particular program uses a particular data item, attribute, stored field, whatever.

In addition the D.B.A. is able to extend the dictionary to include extra information. He is able to do this by specifying his own data descriptors to map any information he is going to input into his own extra set of tables.

9.1.10. Concluding Remarks on the Dictionary.

The dictionary provides a means of storing data describing data. Its simplicity of use is based on the holding of the contents of the dictionary in tables with pointers to:-

- lists or tables of values;
- procedure calls and parameter lists;
- procedure bodies.

The dictionary provides the input for the setting up of directory data which will drive the access to the data base.

9.2. IMPLEMENTATION STRUCTURES.

The basis of the proposed D.B.M.S. design is to separate the control and pointer data from the raw data itself. This is in line with the concept of having a small set of subject files on which to base the various operation systems that use the data base.

9.2.1. The Directory.

The directory record is the 'driving mechanism' which directs the data base manager in its access to the data base. The directory record is derived from the dictionary data.

A directory record is set up for each command/access table combination. The basic feature of the directory record is the mapping required to obtain a data element, included in the access table, from the data base. The method of achieving this mapping is as follows:-

(1) The sub-schema structure table is accessed for each data element described in the particular access table. Using any 'EXCLUDE' entries to remove data elements not required, a full list of data aggregates and/or data items are built up which are maps to the schema.

(2) The sub-schema element table is used to obtain the sub-schema mapping information i.e. the relationship entity and attribute identifiers together with the attribute qualifying information if it exists.

(3) The relationship identifiers are stored in the directory record and will identify which primary key index is to be accessed. The attribute qualifying information will supplement the attribute start address and length to obtain the exact character string required to be accessed.

(4) The schema entity/attribute table is accessed to obtain the start address of the attribute relative to the start of the entity. The necessary key identifiers are also obtained to be stored in the directory record. These identifiers similar to the relationship identifiers will identify the required index tables to be examined.

(5) The attribute table is used to obtain the necessary attribute information for integrity, validity checking etc., as well as obtaining the length of the attribute field.

(6) The entity/field table provides the physical data field mapping from the entity and attribute identifiers. The file structure table provides further information about the records and/or segments to be accessed as well as the physical keys involved.

Taking all the fields together provides a set of data records and/or data segments that must be accessed for a single access table together with:-

- primary key and/or secondary key index identifiers;

- length/displacement information to obtain the exact data required;
- value lists, procedures and parameters required for validity, integrity and access control checking.

The values and ranges that are valid or invalid are resolved from the dictionary details before being set up in the directory record. This means that such as the set list manipulations (JOIN, RESTRICTION etc.) are performed and the resultant set of values are stored in the record. The procedure is stored as an overlayed piece of code (or paged depending on machine architecture) and a call mechanism is set up in the directory record.

9.2.2. Data Files.

A data base file is simply regarded as a named grouping of a set of records and possibly record types. Each record may consist of a number of segments which may consist of further segments. The record and segments are identified by keys. The segments enable larger records, involving possibly repeating groups, to be broken up into more manageable and accessible groupings. The allocation of the file to the storage unit may be machine dependent to some extent (e.g. the file may have to be stored in a contiguous area in whole cylinders).

The page is referred to throughout this thesis as the unit of transfer. The page of a data file contains the following:-

(1) The page contains a table of entries, one entry for each addressable data grouping within the page. The grouping is any segment or record. The contents of each entry are:-

(a) The key value of the segment or record.
(b) The start address of the grouping within the page.

(2) This table is followed by the records/segments in the page. Each record/segment consists of:-

(a) A bit table consisting of a single bit for each field in the grouping. The settings are:-

- if the bit is set the field is present;
- if the bit is unset the field is absent.

This provides an easy method of representing the absence of a field.

(b) The data fields themselves. All fields will be encoded to provide:-

- maximum security;
- maximum packing of data.

The key field of the grouping is automatically absent, being already present in the table at the head of the page. This table is also encoded.

(c) At the position within a segment/record that a new segment commences, a segment table is held containing an entry for each occurrence of the new segment. Each entry can be either:-

- the key value of the segment and the page number where the segment resides, or

- if the particular segment resides in the current page the entry number of the segment within the page list. There is little extra overhead for describing segments in the current page in this way because:-
  - the key is only stored once, in the page list;
  - the whole segment table is also encoded.

(3) A further table is required as a header to the page list. This is to identify the key type in the page list e.g. in the 'ORDERS' file, the 'ORDER' key of the record must be identified from the 'PRODUCT' key of the 'ORDER/PRODUCT' segment within the 'ORDER' record. This is achieved by having a list of the record/segment identifiers (as set in the dictionary) that occur in this page, associated with a short code (1, 2, 3...). This short code is used in the page list to identify the key. The header table will also provide the key length for each record/segment type.

(4) The 'picture' of the record/segment being accessed is held as part of the directory record.

Taking the 'ORDER' file as an example, if a page is accessed with an order number, the order details can be obtained and all products relating to that order. Alternatively an order with details of only one product on that order may be accessed using these structures. The products could be held as segments within the 'ORDER' record.

Very complex record structures could be formed with a single record stretching over several pages, but it would be expected that records would not be too large in size and usually a whole record would fit completely into a page. The larger the record the greater the likelihood of performance degradation.

9.2.3. The Control Area.

The control area is a set of control tables and data which provide all types of central control information required by the D.B.M.S. Examples of information held in this area will be described as and when required. The data in the area can only be read from or written to by the data base manager. It is held permanently in main store.

9.2.4. Primary Index Tables.

Primary index tables simply consist of one entry for each entity in an entity set in the data base. Each entry consist of the following:-

(1) The primary key value.

(2) The page number where the entity data exists. (The entity date is mapped by the directory record into files, records, segments and fields).

(3) At the start of the index page there is a value giving the total number of index entries in the page.

There is not necessarily a primary key index for every primary key. If a primary key consists of a concatenation of 2 other primary keys, the details required for the entity data may be held in 2 separate files. The primary key will obtain the data required by accessing the data through the primary index tables of the 2 constituent primary keys. In the 'ORDER/PRODUCT' primary key the order number is a primary key to a record, but the product number is not a primary key to the 'PARTS' file only a segment key as part of the 'ORDER' record. A primary index could be held for this concatenated key but it is much more likely that access will be required through the order record and that the 'PRODUCT' keys will be held as a segment list within each 'ORDER' record (as described in the previous section).

Access to the primary index is by:-

(1) either a user supplied randomising algorithm;
(2) or, by default, using hash random with prime number division.

It will be the responsibility of the D.B.A. to tune the index layout so that as often as possible (>98%) only 1 access is required to obtain the page address of the data.

When a multiple relationship exists between 2 entities the primary keys for each relationship are either:-

- stored in separate index tables, or,
- stored in one table, but each entry is identified as to which relationship that key belongs. The relationship identifier will be 1, 2, 3... with the interpretation of this code held in the control area. The short code will be associated with the relationship identifier as allocated by the dictionary system. The directory record access mapping will have the relationship identifier which will be used to obtain the short code from the control area prior to accessing the primary key table.

The secondary storage location of each primary index table is held in the control area as a pair of values:-

- a start page number;
- the number of pages in the table.

If primary index relationships are separated, there is a base 'page/number of pages' entry for each relationship.

If the index table is a segment list within a record there is no choice, all keys are held together, with a relationship identifier.

9.2.5. Secondary Index Tables.

Secondary key index tables are the most difficult structures to handle in a D.B.M.S. They are very volatile and if implemented to too great an extent cause a great deal of performance degradation when updating the entity to which the secondary key belongs. The performance aspects are discussed in a later section.

The index is a 2-level structure:-

(1) The first level is a list of all the different secondary key values that are present in the data base. Associated with the key value is a pointer to the list of primary keys that possess that particular secondary key value.

(2) The list of primary keys is the second level. If the list contains more entries than can be held in one page a pointer is set up to point to the page containing the continued list.

The first level page is obtained by a randomising algorithm (default is 'hash random' if the user does not provide his own algorithm). The start page and length of the first level is held within the control area.

If a secondary key is derived by 'WITHIN' clause the number of secondary key values will increase but the length of the primary key list for each secondary key value will almost certainly decrease.

In the 'STATUS-LEVEL' structure, the primary keys are stored as a tree list. Each node of the structure can consist of several primary keys. Each node is identified by a set of branch numbers separated by solidii. If there are 'n' solidii the current node is at the 'n+2' level. A plex structure may be defined by declaring the same primary key value at more than one node.

In addition or as an alternative the structure may be stored in an inverted manner by storing each primary key in turn together with its node value for a tree structure or node values for a plex structure.

The first format would be accessed in a serial top down manner, while the latter format would permit access straight to a particular entry in the structure by randomising on the primary key value.

9.2.6. Implementation of Schema Entity Relationships.

The relationships between entities such as those between 'CUSTOMER' and 'ORDER' entities, 'PRODUCT' and 'ORDER/PRODUCT' entities are implemented in an identical manner to the implementation of the secondary index tables. Using the 'CUSTOMER', 'ORDER' relationship as an example, the location of the 'CUSTOMER' entry is found by randomising at the first level which points to the page address of the second level containing the order numbers of the orders made by that customer. Any 1-to-1, 1-to-n or m-to-n relationship may be defined in this way.

Only inter-file primary key relationships are defined in this way, intra-file primary key relationships, such as those between the 'ORDER' and 'ORDER/PRODUCT' entities, already exist through the segment list.

9.3. D.B.M.S. MECHANICS.

This section describes the following aspects of data base manager mechanics:-

(1) The D.B.M.S. storage areas.

(2) The mechanics of each DML command.

(3) Performance and tuning.

9.3.1. The D.B.M.S. Storage Areas.

The File and Index Buffers.

The pages containing the data and index tables are read into the buffer storage. Similarly, data is written from these buffers to the storage devices.

Index Tables.

If a page containing a part of an index table is read into buffer storage it is decoded into a D.B.M.S. working area. The address of the start of the table is held in the control area. The page number is also held in the control area together with descriptor details describing the format of an index entry. This descriptor is copied from the directory record into the control area and is retained until the index is over-written or written back (if altered) to the secondary storage.

Entity Storage.

When data has been read into a file buffer it is decoded and passed to the D.B.M.S. work area. The data in this work area is grouped into entities. All the entities belonging to a run-unit are linked by pointers with the head-of-list address held in the control area. The descriptor describing the format of each entity is copied into a descriptor table in the control area. The entity data is held in cells of main store but if there is a shortage of space any entity data that is still being used by the run-unit will be written to the temporary file secondary storage area until required by the run-unit.

The sub-schema unlike the schema does not have a separate physical representation and the sub-schema access controls and integrity controls are 'blended' into the schema controls. The merged sub-schema/schema controls are held in the directory record ready to be used on the entity data set(s). The sub-schema controls may be in addition to/or replacements for the schema controls.

Run-unit Working Storage.

The access table describes the data available directly to the run-unit procedures. It is set out in a contiguous area with named data areas identical to the names in the access table (and thereby in the sub-schema). This data description of data base data is included in the run-unit procedure by the dictionary system in a preprocessor run prior to compilation of the procedure. If more than one access table is used by a procedure and a data element is used in more than one access table it is still only declared in the procedure's storage area once.

Both in the entity grouping and run-unit data area, a data element will be stored in the same mode as that described in the physical data description i.e. if a data field is described as binary and an attribute and data element map directly to that data field they too will be described as binary. As an additional facility, the access table data element may have a mode associated with it to permit alteration for the run-unit. However, the entity data will always be held in the mode described by the physical description.

9.3.2. The Mechanics of the DML Commands.

Each command, before doing any data base accessing, reads the directory record to ascertain which functions are to be performed.

FIND

Requests are set up to obtain the primary and/or secondary index tables that are required as indicated by the directory data. The index table may already be in main store; a check is made to discover if the index page required is already in store. This is done by checking the list of page numbers that are in main store, the list is held in the control area. The page numbers are found by using a particular randomising algorithm. The information defining which algorithm is used for which index table is stored in the control area. Requests are set up for the pages required that are not in main store. How each transfer from secondary to main store is performed will be dependent on the particular machine hardware. The particular index is decoded from file buffer to table area and a request for the data page is made if the page is not already in store. Once the data page is read it is decoded from the file buffer and mapped into the entity records under the direction of the directory record. If the FIND involves a complex selection procedure the processing could be considerably more lengthy.

GIVE

The directory record identifies any schema/sub-schema access checking that is to be performed, this may involve accessing a particular procedure or reading access lists from the directory. Any 'empty' attributes may be given a default value and any virtual attributes calculated. To enable these types of procedures to be performed may require more than the data requested by the user to be read into store e.g. to enable the derivation of a virtual attribute. The entity data is mapped into the run-unit's working storage.

PUT

This command first checks which if any attributes have value changes. The changed attributes are checked for validity errors and any integrity checks/adjustments, defined in the directory are performed. Any access control checks specified in the directory record are also performed. If the data is acceptable it is copied back to the entity record. Any elements specified in the sub-schema to have an initial value, have this value written to the corresponding run-units fields ready for the next GIVE command.

STORE

This command transfers the particular entity data to the file buffers encoding the data as it is written back. The original copy of a page may have been thrown out to the temporary file if there is a high-level of activity, the page must first be read back from the temporary file. The data page(s) and possibly index tables (if they have been altered) are available to be written back. There is no need for the page(s) to be written back until the file buffer is required, and so the writing of the page is really performed as a result of requiring to read data e.g. in a 'FIND' request.

REMOVE

There is a list maintained for each user giving details of all entity occurrences removed by this user (i.e. logically deleted). Any lower level entity occurrences that relate to the removed entity also become unavailable. If required, single attribute occurrences may be removed but the default is to remove whole entities.

DELETE

This command removes entity occurrences and any other related entity occurrences at a lower level. The related primary and secondary keys are removed and the particular records or segments are physically removed from the data base files.

TRANSACTION

The data is set up in a transaction file. The date and time at which the transactions are to be actioned are stored in the control area. At the time when the data is to be processed the data that is to be processed is prevented from being used by other transactions until the transaction file processing is complete.

The transaction file may not consist of a set of records, it may contain a procedure which details how a set of entity occurrences is to be updated. The method of locking out the particular data involved in the update will be described later.

REPORT

The date and time that a report is to be produced is held in the control area. Detailsof how the report records are produced will be described in the later section describing 'time consistency'.

9.3.3. Performance and Tuning.

Primary Index Tables.

The D.B.A. should ensure that access to overflow is minimised as much as possible less than 2% of accesses to home pages should require access to overflow pages.

In reading or writing, an entity of data, there should be:

- 1 access to the primary index (sometimes 2);
- 1 access to the data block.

In updating an entity the data block will be read and written causing 3 physical disc accesses (very occasionally 4).

The question arises why not randomise directly to the data page without wasting time and space by going through index tables? The problem is that there is always a trade off between performance and storage. The performance achieved by randomised access is at the expense of unused and wasted storage because to achieve an overflow rate as low as 1% or 2% requires the packing density of the home pages to be no more than 50% or 60%. To waste so much space in the large data files, which will take up in excess of 80% of the

secondary storage, is unacceptable to enable physical data accesses to be reduced by one for every logical access, certainly this places too much emphasis on performance in a commercial transaction system. The storage space used for index tables is only a small fraction of that used for data files and the wastage in space in index pages is acceptable.

In a fully realtime environment, with possibly smaller files and the necessity for the best possible performance, randomising directly to the data page may be necessary. To give extra accessing flexibility with this D.B.M.S. the D.B.A. is able to specify through the physical data description what file organisation is required. The language statements although not defined in Chapter 8 are described in Appendix 3.

Secondary Index Tables.

The purpose of a secondary index is to provide information about the entities which are grouped on the basis of a set of criteria. More often than not the actual data records and segments do not require accessing only their identification (primary key) is required. The accessing requirement is simply to read the secondary index table. This is at least 2 physical accesses increasing proportionally with the number of entities in the grouping. If the entity attributes are required then an extremely large amount of data is involved and the secondary index access overhead relative to the data accessing requirement is small. Retrieval is very efficient.

Updating is not efficient at all. If an attribute value is changed and that attribute is a secondary index, the following accesses are required:-

- 1 access to read the top level secondary index (obtained by randomisation);
- 1 access to read the next level which contains the list of entities belonging to the original grouping;
- 1 access to write this index table back without the entity identifier.

Three accesses are required to remove an entity from a secondary index and similarly 3 accesses are required to add this entity identifier to the new secondary index containing the entities with the new attribute value. Six accesses are required if a secondary index attribute is changed. With 3 accesses required for the primary index read and data read/write the total number of accesses required are:-

$$(6 \times n) + 3$$

where n is the number of secondary index attributes updated. In a D.B.M.S. geared to the best possible performance this number is excessive.

The best solution to the problem is to obtain the best possible performance in updating an entity (where response is usually important) at the expense of retrieval of secondary key information. This is achieved by minimising the use of secondary keys and by obtaining the information by reading every record in the file.

The performance loss would be large but quite possibly acceptable when taking into account the types of processing that are based on secondary key retrieval, these are:-

- batch processing runs (often the runs are off-peak);
- on-line ad-hoc enquiries for which a response of minutes might well be acceptable, permitting the processing to be performed in a background mode to any fast response transaction systems.

Even this solution may not be acceptable for very large files, however a balance between using secondary keys and searching files must be achieved with secondary keys set up, if:-

- the request that uses the secondary key is often required;
- the file is not too large.

Clearly the best solution is only achievable by finding the best balance.

To be able to read a whole file requires a conventional file to be described as a single entity set. The whole file is read by specifying in the 'FIND' access table:-

SELECT WHOLE-FILE;

This is followed by a set of 'GIVE NEXT' commands to pass each record in the file to the program. The reading of the file is performed by reading each page in turn from the disc address from where the file commences. This technique enables conventional pre-data base files to be accessed as data base files with the key of each record described as the primary key. These files can be organised in their original manner with the organisation described within the physical data description.

Batch Processing.

Although the D.B.M.S. proposed appears biased to transaction processing extra facilities are required to process data base files in a batch manner particularly if there is a high file hit rate. The data base files could be accessed in the manner described previously but to aid performance it is possible to take advantage of the fact that more than one record required to be updated may be stored in the same page. This is achieved by describing each primary index table as an entity set in its own right with the page number as an attribute and each entry in the table an entity occurrence. The entity set is mapped as a conventional serial file and each record is read using 'GIVE NEXT'. The primary keys are read into a working file sorted into transaction key sequence and where there is a match of keys the page number is added to the transaction. At the end of this scan the transactions are sorted into page number order and then the transaction file accesses the data base file as a conventional file using the 'DIRECT SERIAL' access method (described in Appendix 3).

The original design described in chapter 8 has been modified in this section to enable data base and pre-data base files to be regarded as conventional files with serial, direct serial, random, sequential and indexed sequential access possible within the D.B.M.S. Not only can data files be regarded as conventional but there is a requirement to regard the index table files as conventional files.

There are other accessing overheads:-

- transfers from the directory file;
- transfers from the temporary file areas when primary storage work space overflows.

Performance of a D.B.M.S. is 80% or 90% about how many secondary storage accesses are required. General facts and figures should be available for the D.B.M.S. to assist the D.B.A. in weighing-up the possible processing options that will give him the best throughput/response. To aid him in performance monitoring, a diagnostic facility is available which identifies the processing path through the data base manager and which pages are physically transferred whether from data or control files. This type of facility is best used in a load test situation when the impact of making a tuning change to the data base manager, or, the addition of a new data base application, must be known before full

implementation. This diagnostic trace may be used in the live situation if a drop in performance or failure occurs. The D.B.A. may wish to switch the trace on for a short space of time. In addition to a trace facility, throughout normal running certain statistics are maintained to identify performance bottle necks.

The D.B.A. has a number of tuning facilities available to him:-

(a) Varying page size. The D.B.A. is able to specify the exact page size required for any file or index table. As a general rule of thumb, he will probably define a small page size for data and a large page size for index tables.

(b) Varying file buffers and D.B.M.S. buffers. The D.B.A. has the facility to change the number of file buffers and the size of the D.B.M.S. buffers where the entity data records and index tables reside. The ratio between the primary index, secondary index and entity data storage areas can be varied to maximise the likelihood of keeping certain types of information in primary storage for the longest possible period.

(c) The placement of files and index tables on secondary storage units is also an important factor in tuning.

(d) The timing of secondary storage garbage collection and reorganisation can affect performance.

9.4. DATA HOUSEKEEPING.

There are 3 areas of housekeeping that must be considered:-

- garbage collection;
- data reorganisation;
- data restructuring.

9.4.1. Garbage Collection.

The D.B.M.S. is able to repack pages to reduce the 'spreading' of data across pages. When there is volatility in a data file, 2 important features occur:-

(1) Gradually more and more free data pages are used.

(2) The original pages contain less and less data as records are deleted.

The more volatile the file the more often the requirement to collect the free space.

There are 2 possible methods, the second of which is used by the proposed D.B.M.S.:-

(1) There is a one-word value for each page of data storage giving the number of 'units' of free space in that page. This method involves too much primary storage for a large set of data bases.

(2) The second method involves far less housekeeping at the time of user access to the data base. The control area keeps a 5-word area for each file (the size of the area can vary). Each word contains the page number and amount of free space available. The 5 words describe the emptiest partially-used pages in the file. When inserting a record the D.B.M.S. will try to place the record in any of the 5 pages and if one of the pages is suitable its free space value is decreased by the length of the record inserted. If none of the 5 pages are large enough, the record is placed in the next totally free page. This page number is also held in the control area. If this page has more free space available after adding the record than any one of the other 5 pages the page with the least amount of space is dropped from the list and the new page takes its place. The number of the next totally free page is stored in the control area. This value is incremented by 1 to give the next free page number. The edge of the file increases and no note is kept in the control area of the effect of deletion of fields, segments or records in existing pages. When a certain page number is reached (the 'page threshold') the D.B.M.S. notifies the central operator of the need for garbage collection to take place. The operator then informs the D.B.M.S. to enter the on-line garbage collection routines. If there is a peak in workload or the operator does not see the request on his monitor the garbage collection will take place automatically when a second 'page threshold' is reached.

There is no garbage collection for primary index tables because space created by deleted entries may be re-used by entries inserted later.

The second level of the secondary index tables are very volatile requiring on-line garbage collection more frequently than normal data. The garbage collection is invoked automatically when the threshold is reached without request from the operator. Summary information is written to a file called the

D.B.M.S. journal which contains information about such events. The data output to the journal includes a brief message and totals of pages in use, pages empty etc. A system journal entry is also output for the data page garbage collection. When a new secondary key value list commences, a new block is allocated to take the new primary key list. When the garbage collector is called in, as many value lists are packed into a page as possible. If a list in a page containing several lists overflows, the list is assigned a new page.

The first level of the secondary index table is randomised and so like the primary index table garbage collection is not relevant.

During garbage collection in data pages there must be locking against user's accessing:-

- the primary index entries involved;
- the data page into which a record is being moved;
- the data page from which a record is being moved.

There must be similar security for secondary index garbage collection.

Garbage collection in on-line mode is a contingency, it is much more preferable to be done, and should always be done at the end of the day when in off-line mode.

9.4.2. Data Reorganisation.

Data reorganisation is very similar to the collection of free space. However reorganisation deals more with storage structure changes and changes to enhance record access performance.

In the reorganisation of data pages, the following changes come under this heading:-

- increase of the file area;
- alteration of file location, or fragmentation of data pages into smaller groups or building of larger page groups to improve access efficiency. Membership or order within groupings may be based on data values within the records.

Reorganisation of primary index tables or the first level of secondary index tables is required when the particular randomising algorithm being used is changed. The basis on which index entries belong to a certain page changes and index entries must be dumped out and restored to different pages. The major reason for this reorganisation is a growth (or decrease) in the number of entries in the table. Such a reorganisation should be foreseen and planned in advance so that the reorganisation can be done in off-line mode at the end of the day.

There are occasions when this reorganisation may be required to be done in an on-line mode. If there is a very volatile entity set there may be very rapid overflow during the course of an on-line day. If the number of overflow pages in use reaches a threshold level the D.B.A. may allocate a contingency block of reserve 'home' pages. The allocation or 'making known' of the extra set of pages to the D.B.M.S. can be done in on-line mode from a terminal using storage control language statements. The area is then immediately initialised and details of the new area are stored in the control area. The randomising procedure is changed to include the extra pages. The reorganisation is requested through the central operator. The whole primary index table is dumped to a work file and all index entries are restored on the basis of the new set of 'home' pages. During this period the whole primary index table is locked out until the reorganisation is complete. Users who are held up because of the locking are sent explanatory messages. This facility is seen more of an emergency measure to handle an unusually large increase in the number of index entries during the course of a day.

If in any of the index or data pages the end of overflow or file area is reached, as an emergency to prevent the D.B.M.S. failing, extra overflow will be allocated in the temporary file.

9.4.3. Data Restructuring.

Data restructuring is the term used when changing any of the descriptions (physical, schema or sub-schema). The restructuring should normally be done off-line, but there are 2 sets of circumstances when on-line restructuring is useful:-

(1) during application development when the flexibility to restructure on-line would be useful;

(2) during D.B.M.S. or system failure, the failure may be corrected by a certain amount of on-line restructuring.

The partial changes that can be made have already been described in the section on dictionary set-up facilities (section 9.1.8.) but to implement the changes, any of the implementation structures may require changing.

Physical data structure changes to the data require dumping and restore. The dump and restore will probably be done on a page by page basis so that as little an amount of data as possible is locked out.

Directory records will be the main structure to be affected by a restructure. To enable to identify which directory records are affected by a partial restructure requires the use of a cross-reference file. For each unit of change (attribute, entity, schema, sub-schema, access table) there is a list of directory records which are affected if that particular unit is changed. For instance, if a particular attribute is changed it can immediately be seen which directory records need changing. Because the means of access to the data base is only through a directory record, all the directory records that require changing have a marker set against them preventing access until the change is made.

Certain restructuring may be impossible to perform in an on-line mode because of the complexity of the change and generally all restructuring unless absolutely necessary will be done off-line with the D.B.M.S. closed down.

9.5. CONSISTENCY.

The speed of processing compared to the speed of secondary storage access, demands that processing resources are shared by users to obtain the best possible performance. The D.B.M.S. is such a resource and must be shared. The description of the concept of several users 'threading' their way through the D.B.M.S. is referred to as 'multi-threading'. How the multi-threading capability is handled varies from machine to machine. In the ICL 2900 series the multi-threading is taken care of by the operating system with each request having its own virtual machine, the D.B.M.S. code shareable, pure and re-entrant. The new operating systems are tending to take care of, to some extent, some of the data management system functions to enhance data consistency and recovery. The proposed D.B.M.S. must provide for full software control in these areas so that if parts of the control mechanism are not required they can be easily removed from that machine version.

9.5.1. Concurrency.

The FIND command has already described the setting up of the data the user requires in entity groupings within the D.B.M.S. system area. The entity data is known only to that particular user. The entities, if they are related are chained together by twin and child pointers i.e. if data is requested concerning a customer, the orders he had made and the products on order the entities would be chained as in figure 9.1.

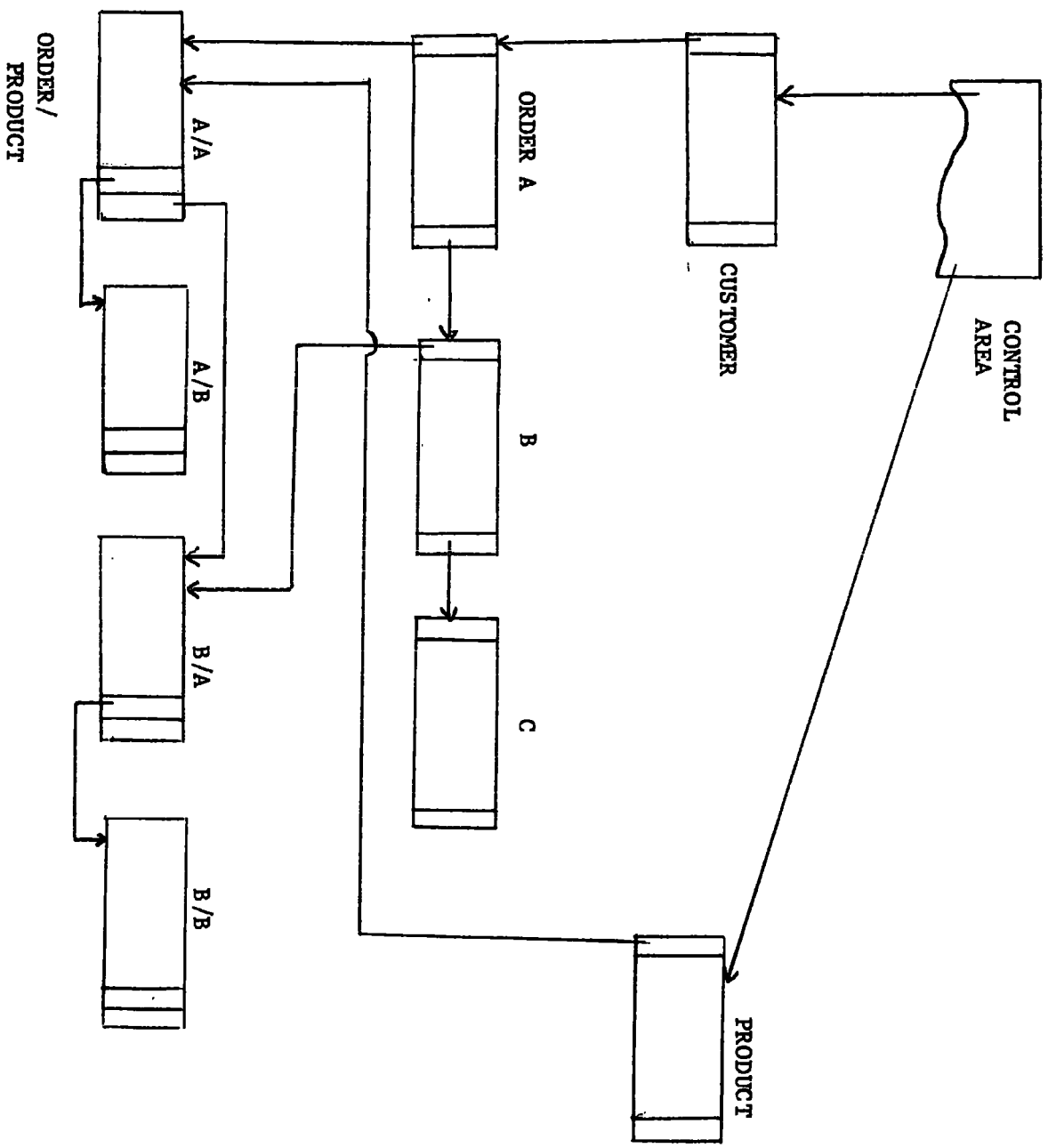


FIGURE 9.1.

The pointers to the customer and product entities are held in the control area. A different user will have his own entity structures.

The following of pointers is not time consuming when in primary storage. Should any part of the structure overflow to the temporary file, an entry is maintained, for every page that contains overflow entity data, to describe the part of the structure that is held in that page.

A control mechanism is required to control the allocation of the physical data to these individual entity structures. The physical data exists in both the file buffers and the secondary storage.

In garbage collection the need to lock out whole pages has been described but to control the building of entities from stored data fields, a far finer degree of resolution is required.

The user must define the boundaries of the success-unit within his run-unit. The user must define:-

- which data elements are to be updated and so require exclusive access;
- which data elements are only to be read and consequently will allow other run-units to read the same data.

The control is maintained by a concurrency control table which contains information about the data that is currently being used. The contents of the table are as follows:-

(1) The data identifier, the record or segment key.

(2) A bit sequence for each data field identifying that field to be in 1 of 3 states:-

- unlocked (bit sequence '0');
- locked for reading (bit sequence '10');
- locked for updating (bit sequence '11').

The position of a data field within a segment or record identifies which part of the bit string applies to the particular data field.

(3) The head of queue address and length of queue of requests waiting for the freeing of a particular data field.

The contents of the 'request slip' is as follows:-

(1) The run-unit making the request.

(2) The record/segment identifier making the request.

(3) The list of data fields required, each field is identified by position within record/segment.

In addition, for each run-unit requesting a set of data fields, a table is maintained, in exactly the same format as the concurrency control table (excluding 'request slip' entries), containing those data fields that have been locked for that run-unit.

The original design method attempted to prevent deadlock by preclaiming all data resources. This is putting an impossible burden on the way data can be processed. Often the need for extra data, results from values obtained from previously requested data.

The strategy proposed is to permit data to be locked on request in any number of events, and that all data should be unlocked as soon as possible.

In the access table for the 'FIND' command, the data elements that may be updated must be accompanied by the word 'UPDATE' to indicate which data elements may be updated. During the 'FIND' request if all data fields are available the concurrency control table and the run-unit's own resource table are updated and the D.B.M.S. goes away to find the record. Subsequent 'FIND' commands follow the same procedure. The 'STORE' command releases the locks from those data fields that are mapped by the data elements in the 'STORE' command access table by updating the two tables mentioned above.

From the details held in the request slips and the run-unit resource table, the D.B.M.S. can identify whether there is a deadlock situation (i.e. A waiting for B's resources waiting for C's resources waiting for A's resources). In such a situation the run-unit processing must back-out to the beginning of the current success-unit. The boundary between success-units is defined by the 'UNLOCK' command. This command has no access table parameter and its effect is to release

all the run-unit's data resources (including entity data). The command also causes the run-unit's local work space and terminal related data to be dumped to enable restart to be made from that point.

If a run-unit can release resources by using the 'STORE' command as suggested, if the run-unit should still possess some of the data resources and then submit another 'FIND' command within the same success-unit which then becomes deadlocked, then to restart at the start of the current success-unit (i.e. when the last 'UNLOCK' was actioned) would mean an inconsistent data base because some of the data had already been made available to other run-units by the 'STORE' command. The data base must abort any run-unit that uses a 'FIND' after a 'STORE' within the same success-unit (before the next 'UNLOCK'). To use the 'PUT' command is permissible because the data is not available to other run-units and it can be retrieved by a 'GIVE' command. The 'STORE' command must have the capability in such circumstance to write to the data base all occurrences of an entity. In the 'order' example when a new order is placed, each of the products must have their 'quantity-on-hand' values updated. A single 'STORE' command when executed will write all the product details away to the data base. The form of the 'STORE' command is:-

STORE ALL access-table-name;

As an alternative if all the data that is required is known at a point in time then it is possible to state the data requirement in one statement, reducing the chance of deadlock by preclaiming as much data that is known to be required at that point in time. This is achieved by using a 'KEYLIST' which addresses an area in the run-unit's local store containing the keys that are required for access. The area for the list is allocated automatically to the run-unit's work space by the data dictionary prior to compilation. In this way all the product details for an order can be claimed together, and after processing for a single product is complete that product's details can be written to the data base and made available as soon as the product is updated. In the cases where the data requirement is known it is better to have one large 'FIND' request and several 'STORE' requests than several 'FIND' requests and one 'STORE'. The format of the 'KEY LIST' command in the 'FIND' access-table is:-

KEY LIST list-name;

An added advantage to the single 'FIND' method is that the data will be accessed in key-list order. The run-unit can continue processing after the 'FIND' up to the first 'GIVE' at which point the run-unit will be suspended until the first logical record becomes available.

Each 'GIVE' when used with a 'FIND' which uses a key list can take the form 'GIVE NEXT'. Only if the next key required is out of the 'KEY LIST' sequence need a key value be given. In the same way 'PUT NEXT' can be used.

The way that accessing to the data base is structured will have a great effect on how good a performance is achieved. The 'golden rule' is that success-units should be as short as possible.

9.5.2. Time Consistency.

Time consistency is necessary to provide data that is consistent at a certain date and time. This may be essential, in the production of certain reports. A report can be requested by a 'REPORT' command. The access table for this command states the date and time when the report is required as well as the logical data that goes to make up the report.

When the date/time occurs this triggers the reading of the data required and the writing of the data to a work file. During this process other run-units may have changed data that is required for the report. When all the report data has been collected into the work file the 'before image' file is examined for any data that has been accessed by other run-units since the time of the report. Any data that has been updated is over-written by the original data in the 'before image' security file.

A report program utility can use the data as input to produce the report in the structure and format that is required.

9.5.3. Audit Trails.

The audit trail documents the data that is read, if it has been changed how it has been changed, who has read or changed the data, when did it happen? The audit trail is documenting the logical data processing.

For each 'GIVE' and each 'PUT' an audit trail record is written to the audit trail file describing:-

- the data that is being read or written;
- the entity data structure that is being used;
- the integrity, validity and access control checks; performed on the data and the results of those checks;
- which terminal, user, program accessed the data;
- the date and time that the access took place.

9.6. RECOVERY AND RESTART.

There are 3 main areas of failure, each one requiring different techniques for recovery:-

(1) file failure;
(2) system failure;
(3) run-unit failure.

9.6.1. File Failure.

Data base files are becoming larger and larger and the problems associated with keeping an increasing volume of data secure become greater.

With very large data base files it may be impossible to dump the files at the end of the day which is a normal practice for present-day T.P. systems. The D.B.M.S. may provide both an on-line T.P. system during the day and a batch environment during most of the remaining 24 hours. The D.B.M.S. must provide facilities to dump sections of the files at intervals while the D.B.M.S. is providing a user service. It is likely that dumping will have the lowest priority level of any D.B.M.S. task and it will only function when there is not a peak load through the D.B.M.S. The dumped files are dumped on the basis of their updating activity and they form the basis of file recovery. The dump files should be held on disc and on tape, the dump tape should be copied to another tape and one of the pair should be held away from the installation.

If data is written to the data base with a 'STORE' command or an index entry in an index table is changed, the relevant page is written as an 'after image'

record to an update log file. If a file failure occurs the particular file or part of the file is restored automatically from the dump file and any relevant 'after image' records written after the dump was taken are superimposed on the restored file.

9.6.2. System Failure.

The significance of this type of failure is that the internal main store contents are lost. The failure may be hardware, operating system or D.B.M.S.

The major problem in such a situation is the need to synchronise the different run-units. In an on-line mode, each run-unit starts with the input of a transaction or request and terminates with an output reply. At any point in time the run-units in the machine are at different stages of completion.

The D.B.M.S. provides the facility to the D.B.A. to provide a value to the D.B.M.S. indicating how many messages are to be permitted between the taking of checkpoints. When a checkpoint is called into operation, all transactions are locked out of the machine until cleared of all transaction run-units. Any batch or background on-line run-units are temporarily suspended (the method of suspension will be dependant on the particular machine but must be done at the end of a success-unit). A copy of the control area is written to secondary storage together with terminal data consisting of data passed between transactions (run-units) in a multi-transaction conversation. The checkpoint number which is held in the control area is incremented by 1. All file buffers, index tables and entity data must also be copied as part of the checkpoint. All processing is allowed to continue.

A warning message is output to each terminal when the checkpoint is in progress. If data is sent from a terminal an explanatory message is returned and the transaction data is held ready for the resumption of processing.

All incoming transactions are logged. The transaction log record details include:-

- the current checkpoint serial number;
- the transaction header details containing date, time, terminal number, message serial number etc.;
- the transaction data itself.

'Before image' records of all data and index tables read into main store are written to the update log file. The log record also contains the current checkpoint number.

When a system failure occurs:-

- the current system checkpoint (by default) or if desired a previous system checkpoint is loaded into the machine;

- the data base is de-updated with the 'before image' records using the checkpoint number to take the data base back to the checkpoint data base state;

- the transactions with the current checkpoint number are reprocessed in accordance with their terminal number and transaction time. This brings the system back to an up-to-date state.

If there is a permanent fault in the operating system, D.B.M.S. or in a particular run-unit which caused the original failure, there is no reason why such a failure may not re-occur. If this happens recovery will have to be made manually by the D.B.A. Details of the facilities provided to help the D.B.A. are described later.

9.6.3. Application Failure.

Failure of a run-unit is one of the most difficult types of failure to correct. It may cause D.B.M.S. failure but it is more likely to go without notice until either a user at a terminal or a later run-unit discovers the error.

This type of error more than any other requires manual intervention by the D.B.A. to:-

(a) be able to diagnose the fault;

(b) to correct the fault.

A diagnostic trace facility is provided to identify the path taken through the data base manager. This should highlight where a particular failure is occurring.

Any application which is in error must be 'locked out', preventing its further use, and also corrupted data must be available to be examined and corrected by the D.B.A. The directory record cross reference file, already described, enables different parts of the different data description levels to be 'locked out'. Similarly, any page in any file can be locked from use until inspection and correction is done.

9.6.4. Error Handling.

Any of the validity, integrity and access control mechanisms can produce numerous errors. The errors that can occur can be separated into 2 groups as follows:-

- errors which are handled by the run-unit itself i.e. errors returned from DML commands;
- errors which cause the run-unit to abort.

The first type of error causes a reply word to be set to a particular error value. The set of errors, their description and where each individual error is used is described in the data dictionary. Prior to compilation of user code, during the dictionary preprocessor run, a list of errors and their meaning are included on the compilation listing giving the possible replies for each command and access table. This provides important documentation to assist the programmer in his task of testing for error replies.

The second type of error causes the run-unit to abort. It has already been described in the previous section. It will usually involve support staff to be called in to investigate.

9.6.5. Prevention Better than Cure.

The D.B.M.S. provides the facility to check through processing by using a user code trace facility. It can be used in the early stages of testing to identify if there are any incompatibilities between DML command and access table sequences. Its main purpose is to provide documentation to describe the different data elements, entities, attributes, keys and fields that are being

accessed, as well as the integrity, validity and access control checks that are performed. It provides documentation as to the data base processing being performed by user code.

In the testing and parallel running using a live data base (described in chapter 7), the live data base is read (provided access control permits access) but any data written in either test mode is written to the temporary file as though it were the real data base. All secondary index tables and primary index tables that point to this written data are kept separate from the live tables in the temporary file for the duration of the test or parallel run.

10. CONCLUSION.

This final chapter describes why the proposed design in chapters 8 and 9 is well suited to the future growth of the information technology industry.

10.1. A REVIEW OF THE DESIGN.

The basis of the design is to look at information within an organisation as it exists, not as it is currently used by the data processing department or as it is expected to be used in the future, not even as it is stored on the computing system.

Data exists as a reality, an organisation needs only to be aware of a subset of that data for it to function at the optimum level of efficiency, performance and cost effectiveness. Data, of itself, will not bring about effectiveness, it is the way that different parts of the organisation view and use the data. This 'appearance' or interpretation of the data may vary enormously, the requirement is to provide as much flexibility in viewing the data as possible with the best possible performance in fulfilling this aim. The view and use of data may be regarded as central to the day-to-day operation of an organisation but the data viewed (and hence analysed) over a period of time can provide the basis for future organisational planning.

10.2. FLEXIBLE IMPLEMENTATION.

MODULAR.

The user of any complex software system must be able to tailor the system to his own specific needs. The design should be implementable on anything from a small business desk-top machine to a large mainframe. The user must be able to fit a reduced set of facilities to a small number of resources.
To achieve the best implementation flexibility, with the possibility of using different machines, means that the basic philosophy of implementation is that it should, where possible, be machine independent, achieving the maximum portability across a machine range and mongst several different manufacturers. The D.B.M.S. compiles to a machine independent target language which involves the writing of an interpreter to convert the target language to machine code.

The concept of the descriptor is important. Each D.B.M.S. implementation structure can be described and not assumed so that all access to any structure (e.g. a primary index table) is made through a descriptor. The format of such a structure can then be changed, its accompanying descriptor is changed and the need to change D.B.M.S. code is much reduced.

EXTENSIBILITY.

Modularity implies ease of change and ease of extension to existing facilities.

The user is able to add to his own facilities with the use of his own procedures at each level of data description.

RECONFIGURATION.

As well as modularity aiding changes to user requirements, modularity gives ease of reconfiguration, enabling changes to a user's hardware system to cause minimum impact on changes to a D.B.M.S. Ease of reconfiguration also includes ease of D.B.M.S. software development to meet changing hardware technology.

10.3. COMPARING DATA MANAGEMENT PHILOSOPHIES.

Not only is flexibility of viewing (accessing) data a significant part of the design philosophy but the developing processing options will call upon data management to provide a new degree of flexibility and cost-effective performance. The cheap microprossor extends options at the small end of the hardware spectrum while the distributed array processor brings new levels of processing power.

With this and other factors in mind, it is worth comparing the design philosophy proposed in this thesis with the commercial D.B.M.S. of the present day. One of the most recent implementations of data management hardware and software is the ICL 2900 Series using I.D.M.S. The 2900 hardware and operating system 'blends' with I.D.M.S. to provide a 'bridge' from conventional file systems to a data oriented approach. It provides a functional approach to easing a user into data base management. Data dictionary facilities coupled with a data description language, providing a limited amount of data independence and access control, guide the user to think of his data as a resource to be documented rather than the object of processing. Added to this, I.D.M.S. provides an access method, to enable related data to be structured using the CODASYL concept of sets, and extra security. It is the functional approach

providing data base facilities for those systems that have a complex structuring and accessing requirement while permitting conventional files to be used by other less demanding systems.

The proposed design in this thesis is a strategic approach; data relating to the enterprise rather than data relating to the way it is processed is the central theme. A unified approach which provides facilities to support conventional files efficiently. The philosophy is that all new developments are under the control of the D.B.M.S. providing the basis for an information systems' service. ICL seem to be placing emphasis at the moment on carrying users gently with them into the world of data base management but ICL and all manufacturers must replace or envelope their current D.B.M.S. packages to provide a strategic approach to data management, concentrating on providing an information service rather than a means of accessing data.

It is important to be able to access conventional files within the D.B.M.S. and to have enough flexibility in mode of operation to give the best possible service to the user.

10.4. A D.B.M.S. FOR THE FUTURE.

Storage Developments.

The D.B.M.S. described in chapters 8 and 9 was described in the context of today's technology, but the value of the proposals rest in their implementation using future technology. The main reason for this suitability is the clear identification of categories of data under the control of the D.B.M.S. These categories are defined by the likelihood of access. The categories of data sets are in decreasing order of expected access:-

(1) The D.B.M.S. control file. Even under the current design specification this control data is held in primary storage from the time the D.B.M.S. is loaded.

(2) The temporary file.

(3) The directory records and primary index tables.

(4) Secondary index tables.

(5) Current operational data.

(6) Historic data for management information purposes.

The impact of storage technology developments could be as follows:-

- Categories (2) and (3) together with category (1) would be held in electronic-only cheap storage (100 megabytes plus).

- Categories (4) and (5) held on secondary disc storage. The secondary index tables, or the first level of index may even be held in electronic-only storage.

- Category (6) instead of being held off-line on magnetic tapes would be available on-line on a mass storage device.

- A further level of storage which is starting to be used more and more at the present day is storage on microfiche. This is very useful for storage of historic and reference information not directly used in an information system. Summary data, based on the historic data may be held on-line as part of the information system.

It is more than likely that accessing the data base would be improved by several hundred percent because the lower category groups contain the data most often accessed.

Processing Developments.

The development of relatively cheap processing capabilities with increasing use of micro-processors mean that the use of hardware for searching will steadily increase, useful in areas such as index searching.

The structure of the proposed D.B.M.S. with the emphasis on a single directory record directing the processing requiredto provide the data to the user, means that the processing functions which are currently implemented by software routines could be implemented by individual micro-processors performing each individual function under the direction of the directory record.

The D.B.M.S. is also geared to future hardware searching based on relational tables. The structures of data and data descriptors (directory records, index descriptors) will tend to 'merge', as the index tables, their descriptors and the directory records which drive accessing through the index tables are held as relations probably implemented as triads.

The D.B.M.S. would support a future management information system with a very powerful, high level interrogation language. The language statements would

immediately generate directory records which probably would be physically structured as triads. Any access control checking would be performed as the directory record is set up. This would involve a complex access control system providing flexibility but not at the expense of security.

It would be possible to set up a limited information system capability with current technology. The important aspects of the system would be:-

(1) The use of a powerful interrogation language generating directory records interactively with a secure access control system as described above.

(2) The overnight generation (update) of a great number of secondary level index tables. The tables would be constructed for the information system only and would not be read or updated by operation systems. This would keep operation system overheads to a minimum.

(3) The capability to 'break down' operation data bases to form information data bases containing mainly summary data. The operation data base files may be used to update existing information system data base files overnight.

(4) An additional facility provided by the D.B.M.S. and not defined earlier, that is to enable data physically deleted from the operation data base (e.g. orders that have been sent out and payment received) to be written to a history file automatically by indication through the DELETE command. This history file can be used to update the information system data base.

Such an information system would be based on 24 hour old data but such a limited information system facility would provide a limited and beneficial service without affecting the overall performance of operation systems.

10.5. FINAL REMARKS.

In conclusion, the proposed design is an answer to the GUIDE/SHARE requirements with certain predominant features:-

(1) Data independence is enhanced with the use of access tables.

(2) The programming interface is simple.

(3) A great many accessing options are available.

(4) Although a flexible system, emphasis is placed on performance.

(5) Users can extend the D.B.M.S. with 'own procedures' at various data description levels.

(6) The importance of extensibility and portability is emphasised. The D.B.M.S. access to its own internal structures is through descriptors making changes to these structures easier to implement and making transfer to a different machine easier.

(7) Conventional files from existing non-data base applications can be easily included under the control of the D.B.M.S. requiring only small modification of existing programs.

(8) The D.B.M.S. can support a limited information system.

(9) Above all, the D.B.M.S. is designed with future technological developments in mind.

A P P E N D I C E S

A P P E N D I X 1.

A METHOD OF DESCRIBING DATA STRUCTURES.

Bachman Diagrams can be used to relate any type of object to another object to permit a structure to be formed. Figures 1 and 2 describe the logical mapping of a file and the physical mapping of a disc store respectively.

Figure 1 is describing that for a particular file there are MANY records and for each record there are MANY fields. When the file is loaded on a storage device there is a 1-to-1 relationship between the file and that storage device, however when the file is unloaded the relationship is broken. The dashed line represents the fact that the relationship can be broken.

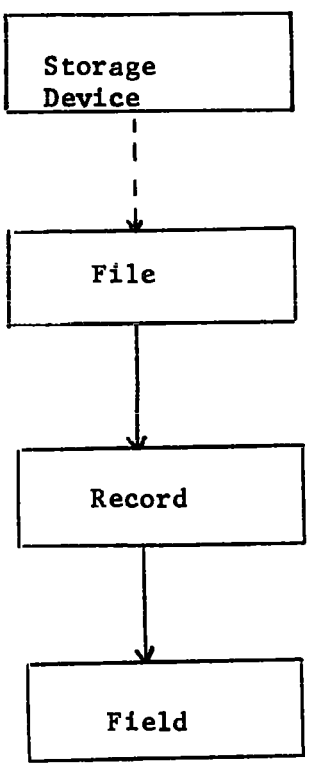


FIGURE 1.

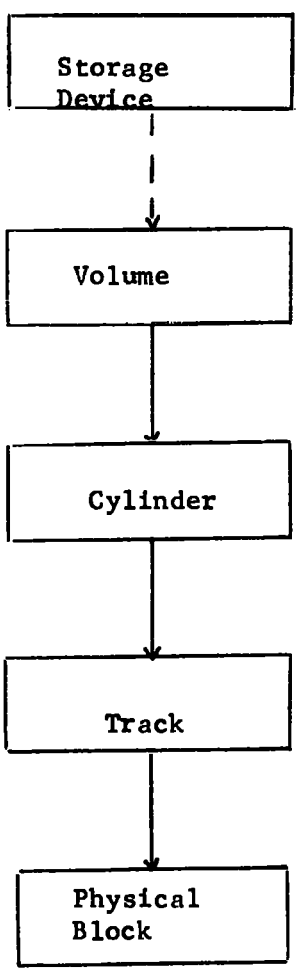


FIGURE 2.

Figure 3 illustrates the set of structures described in chapter 7.

The sub-schema structure is relatively complex providing plenty of flexibility for defining relationships. The data item has 3 pointers to it meaning that there is a 1-to-n relationship between:-

- a record and data items;
- a data aggregate and data items;
- a vector and data items.

The aggregate and vector boxes have 2-way lines indicating that an aggregate can be part of a vector and a vector can be part of an aggregate.

In the schema structure, the entity has associated with it (is identified by) a primary key. There is a 1-to-1 relationship between the entity and the primary key. This is symbolised by the two boxes being joined by a dashed line.

In the physical and storage structures the page is defined as the unit of transfer. It is the means of linking the physical data structure to the storage structure. In this thesis the page is regarded as a logical concept while in some machines the page is regarded physically as part of the storage structure and in this latter case the page has a fixed size. An example of the physical page is the fixed head disc page on the ICL 2900 machine range. The storage structure will vary dependent upon the physical device characteristics.

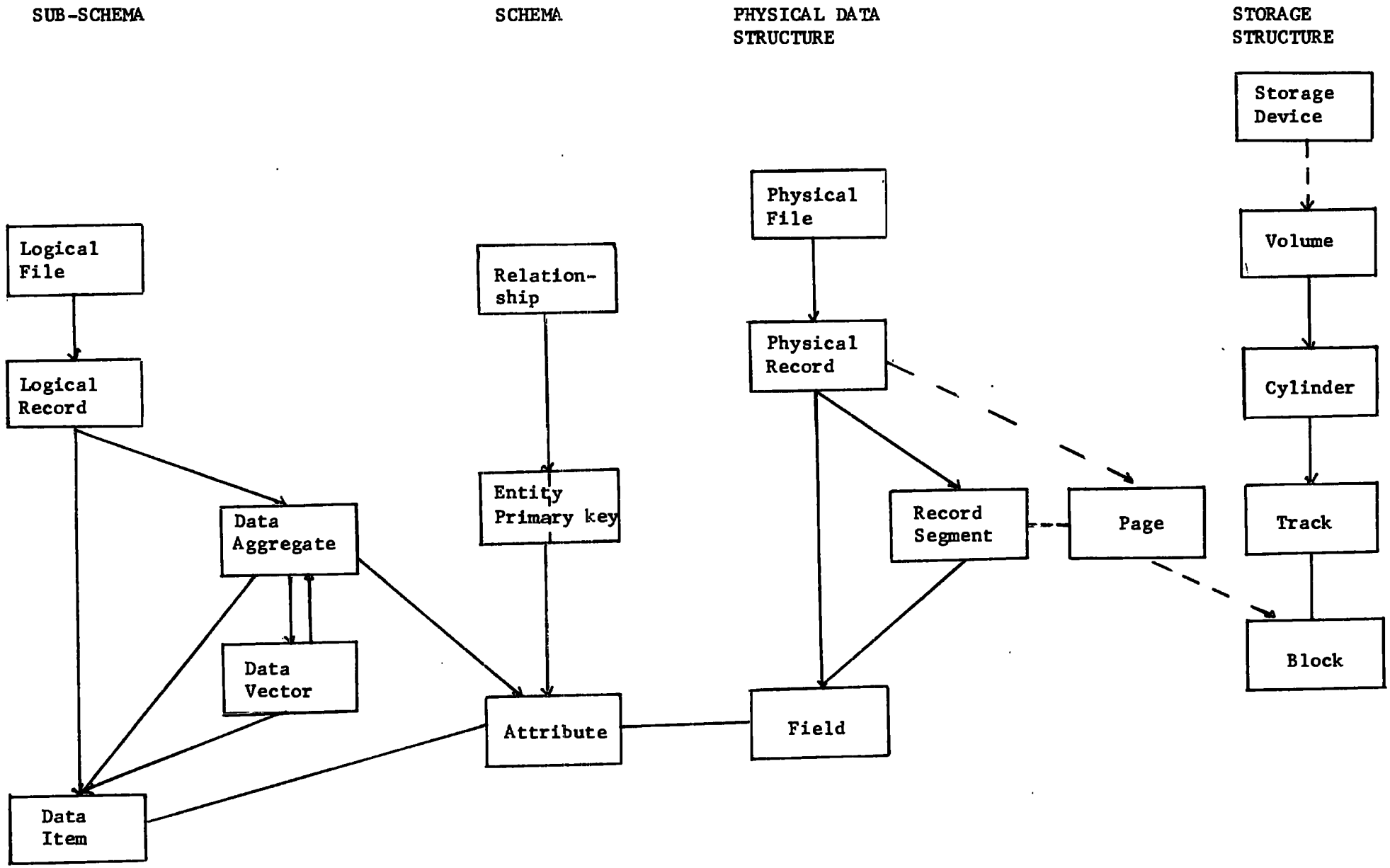

SUB-SCHEMA
SCHEMA
PHYSICAL DATA STRUCTURE
STORAGE STRUCTURE
Logical File
Logical Record
Data Aggregate
Data Vector
Data Item
Relation-ship
Entity Primary key
Attribute
Physical File
Physical Record
Record Segment
Field
Page
Storage Device
Volume
Cylinder
Track
Block

FIGURE 3.

A P P E N D I X 2.

A SYSTEM TO SUPPORT VOLATILE FILES.

There is a requirement in any TP or data base system to be able to use temporary disc file storage. The advantages of use of such a file are:-

(1) Storage areas are easily allocated/de-allocated.

(2) Temporary data is easily read and written.

The basic aim is to store the data in a convenient manner with the emphasis on quick access and security against failure in a multi-terminal environment.

The temporary file consists of a set of fixed-sized pages, the first of which contains a bit map. Each bit in the map refers to a page in the file. The first bit refers to page 2, the $n^{th.}$ bit refers to the page n+1. The meaning of the two possible bit values are:-

(1) Value '0' - the page is not allocated.

(2) Value '1' - the page is allocated.

The remainder of the first page may contain general control information such as the length of record keys.

It is assumed that there is a control record for each data base run-unit or T.P. terminal in which there is held control information pertaining to the particular data base run-unit or T.P. terminal conversation. The characteristic of this control record is that it is read into main store from a control file at the commencement of each run-unit processing or T.P. conversation before any other processing is performed, and the record is written back to the control file at the completion of this processing.

The control record lists all the pages that have been allocated to that run-unit. The structure of each list entry is as follows:-

(1) The key of the slot that is stored on the file.

(2) The page number where the data resides.

USER ACCESSING.

Read Slot.

(1) The particular control record access list is searched for the slot required.

(2) When the correct entry is found the address (page number) of the data is obtained.

(3) If the key is not found an error reply is sent to the user.

(4) If the key is found the particular page is read. The page may not require to be read if the page required is already in primary storage.

Write Slot.

(1) The control record access list is searched to ensure that a slot of the same name for that run-unit does not already exist. If there is already a slot in existence, an error is returned to the run-unit.

(2) The first bit in the bit map that is found unset is set. The number of the bit entry set is the page number to be allocated for the writing of the slot.

(3) The slot is written to the page.

(4) The slot key is set in the control record access list as the last entry in the list. The first word at the head of the list gives the number of entries in the list. This is incremented by '1' giving the new list entry where the key is to be stored.

Update Slot.

(1) The control record access list is searched to ensure the particular slot is on the list (i.e. the slot already exists). If the slot name is not found an error reply is sent to the run-unit that made the request.

(2) The steps (2), (3) and (4) defined for writing a slot are used to add the new updated version of the slot. In effect there are 2 versions of the same slot both allocated on the access list with page space allocated on the bit map. The latest version of the slot is nearest the end of the access list.

Delete Slot.

(1) The access list is searched and all versions of the slot have their entries removed (i.e. the keys are removed but the page numbers remain).

(2) The page numbers of the deleted slot version are used to zeroise the correct bit entries in the map (this is performed later during housekeeping).

In reading, writing and updating slots there is one physical disc access involved. To delete a slot requires no disc access at all. In addition each run-unit must read the access list at the start of processing and write the list back at the end of processing, remembering that this extra read and write would have to be performed anyway owing to the need to have available to the run-unit other control information. The bit map is permanently in store and is manipulated by all run-units.

The design will provide a fair degree of user flexibility:-

(1) It will be possible to vary the size of the transfer unit to best suite the needs of the user although to aid simplicity the size chosen should apply to all transfer units within the file. Of course the choice may in some ways be restricted by the hardware storage device characteristics.

(2) The slot key length should be able to be varied to suite the user although once again, the choice should apply to all slots.

SECURITY.

The security of the temporary file in a multi-user multithreading environment is based on each run-unit's access list.

In a transaction processing environment, to which this particular facility strongly relates, the first processing to be performed is to read the access list and the last processing performed is to write the list back. If the list has been successfully written back the slots belonging to that user are secure at that point in time.

If any failure occurs either before or during the writing of the list, the file will be recovered to the start of the current transaction and it is assumed that any D.B.M.S. in use at the time will at least have a similar capability.

When a failure occurs all primary storage contents are lost and the disc version of the access list which is the secure list state at the start of the failed transaction provides all the information necessary for recovery. All the different access lists are used in turn to reconstitute the bit map using the

page numbers on the lists. When this process is complete the temporary file's state is recovered ready for reprocessing or retransmission of the transactions which were being processed at the time of the failure.

This is the main reason why multiple copies of the same slot are maintained with no overwriting or removal of slot data by the update or delete functions. The original start-of-transaction copy of data is still available.

At the end of the run-unit a certain amount of housekeeping must be performed before the access list is finally written back to secondary storage. This housekeeping involves 'compressing' the list by removing all but the latest version of each slot and at the same time clearing the bits in the map associated with earlier versions of a slot.

After the tidying process the access list is immediately written to secondary storage. From the start of the tidying process to the successful completion of the access list write, the bit map must be locked preventing access from any other run-unit. The length of time the bit map is locked may be shortened by tidying the access list before doing the accompanying bit map update, so enabling the tidying of the access list before the locking of the bit map, the writing of the tidied list must nevertheless be the last process at the end of which the lock may be removed.

D.B.M.S. USE OF TEMPORARY FILE.

In chapter 8 the need was described for the conceptual D.B.M.S. to use a temporary file. The file could be used:-

- as a very large D.B.M.S. work area;
- to support the collection of a large amount of data base data into a sub-schema logical file before processing, probably in batch mode.
- to enable the use of the SAVE and RESTORE DML commands;
- to enable the use of the TRANSACTION and REPORT DML commands to hold the transaction and report details. The report data is written to a work file;
- the temporary file holds the test data base when a test or parallel run is performed;
- the temporary file is used as an overflow area for file buffer data and entity data which is still in use.

Unlike the normal user, the D.B.M.S. can chain pages together with pointers from one page to the next page. Both chaining and indexing can be used for different parts of the temporary file. The choice between using chaining or indexing is made by the D.B.A. and is based on how the data is to be accessed. If at all possible chaining is preferred as it cuts down the index size which is held in main store.

A P P E N D I X 3.

LANGUAGE SYNTAX FOR DATA DESCRIPTION LANGUAGES.

Words in Capital Letters	-	these are keywords.
Underlined Keywords	-	these words are necessary in the context within which they exist.
{ }	-	one of the possible alternatives within these parentheses must be present.
[ ]	-	the contents within these parentheses are optional, only one of the alternatives may be present, otherwise none of the alternatives are necessary.

Schema Data Description Language.

```
SCHEMA          schema-name;

ENTITY          entity-name;

ATTRIBUTE       attribute-name

                PICTURE (string of characters 9, X, (, ), A, +);

[ { VALID   }   VALUE = literal␣literal.....;
  { INVALID }

  { VALID   }   RANGE = literal - literal;
  { INVALID }

  { VALID   }   [ RESTRICTION ]          LIST  list-name;
  { INVALID }   [ JOIN        ]
                [ COMPOSITION ]
                [ SEQUENCE    ]

  VALIDATION    PROCEDURE procedure-name[param1, param2,..., paramn];

[ CODING      { PROCEDURE procedure-name[param1, param2,..., paramn]; }  ]
              { TABLE     table-name;                                   }

[ VIRTUAL PROCEDURE procedure-name[param1, param2,..., paramn]; ]

[ DEFAULT = literal; ]

  ATTRIBUTE END;

  PRIMARY KEY key-name = entity-name1. attribute-name1[(a,b)][:...
                         entity-name2. attribute-name2[(c,d)] :...
                                       :
                         entity-namen. attribute-namen[(y,z)]];

[ SECONDARY KEY key-name = entity-name1. attribute-name1[(a,b)][ :...
                           entity-name2. attribute-name2[(c,d)] :...
                                         :
                           entity-namen. attribute-namen[(y,z)]];

    [ [ NUMERIC      ]   VALUE { TABLE table-name;                    }  ]
    [ [ ALPHABETIC   ]         { RANGE FROM literal BY                }  ]
    [ [ ALPHANUMERIC ]         {       numeric-literal TO literal;    }  ]
```

[SECONDARY KEY key-name = PROCEDURE procedure-name[param$_1$, param$_2$,...,param$_n$]

[[NUMERIC / ALPHABETIC / ALPHANUMERIC] VALUE { TABLE etc. ; / RANGE etc. ; }]

SECONDARY KEY key-name$_1$ = key-name$_2$ WITHIN key-name$_3$[WITHIN...
...WITHIN key-name$_n$]

[[NUMERIC / ALPHABETIC / ALPHANUMERIC] VALUE { TABLE etc. ; / RANGE etc. ; }]

SECONDARY KEY key-name = STATUS-LEVEL;]

[INTEGRITY PROCEDURE procedure-name[param$_1$, param$_2$,..., param$_n$];]

[PRIVACY PROCEDURE procedure-name[param$_1$, param$_2$,..., param$_n$];]

ENTITY END;

[RELATION;

relationship-name$_1$?[...? relationship-name$_n$?]entity-name$_1$ { = / > / < > } entity-name$_2$;

RELATION END;]

[RESTRICTION;

relationship-name$_1$?...? relationship-name$_n$? entity-name$_1$;

RESTRICTION END;]

SCHEMA END;

Sub-schema Data Description Language.

```
SUB-SCHEMA  sub-schema-name;
[ FILE  file-name; ]
RECORD  record-name;

{ ITEM item-name =[relationship name1?...? relationship-namen?]
                  entity-namea. attribute-nameb[(x,y)][:
                                  ⋮
                  [relationship-name1?...? relationship-namen?]
                  entity-namec. attribute-named[(r,s)]] ;

  ITEM  item-name PICTURE picture-value; (using X,A,9,(,),+,-)
  ITEM  item-name = STATUS-LEVEL; }

[ AGGREGATE data-element-name
      [=[relationship-name1?...? relationship-namen? ]
        entity-namea. attribute-nameb [(x,y)][ :
                  ⋮
        [relationship-name1?...? relationship-namen?]
        entity-namec. attribute-named [(r,s)] ]  ] ;
[ AGGREGATE END; ]

[ VECTOR  data-element-name; ]

[ VECTOR END; ]
```

Statements applying to items or aggregates:-

```
[
        ACCESS LEVEL  data-name;

        ACCESS TABLE  table-name;

{ INCLUDE } [ OUTPUT      [ READ   ]  ]
{ EXCLUDE }                 [ WRITE  ]
                            [ REMOVE ]
           data-name        [ DELETE ]

 [ TERMINAL ] [ LIST  list-name                                          ]
 [ USER     ] [ PROCEDURE proc-name [ param1, param2,..., paramn ]       ] ;
]

[ INTEGRITY PROCEDURE procedure-name [ param1, param2,..., paramn ] ; ]

   [ VALUE   =  literal; ]

   [ DEFAULT =  literal; ]

   [ INITIAL =  literal; ]
```

Physical Data Description Language.

```
[ DIRECT SERIAL    ]                [ PRIMARY INDEX   ] FILE  file-name;
[ SERIAL           ]                [ SECONDARY INDEX ]
[ SEQUENTIAL       ]
[ INDEXED SEQUENTIAL ]
[ RANDOM           ]

   RECORD  record-name;

[ SEGMENT  segment-name; ]

[ SEGMENT END; ]
```

```
[ KEY ] FIELD field-name = [entity-name1.] attribute-name1[(a,b)][ :
                                    ⋮
                           [entity-name2.] attribute-name2[(c,d)]][,
                           [entity-name3.] attribute-name3[(e,f)][ :
                                    ⋮
                           [entity-name4.] attribute-name4[(g,h)]]]

                    [ MODE { BINARY    } ]
                           { UNPACKED  }   ;
                           { CHARACTER }
                    [                  ]

[ CODING [ BINARY    ] PROCEDURE procedure-name [param1, param2,..., paramn] ;
         [ UNPACKED  ]
         [ CHARACTER ]

  CODING NONE;

  CODING TABLE table-name;                                                  ]
```

Data Manipulation Language.

```
FIND        access-table-name;
REMOVE      access-table-name;
DELETE      access-table-name;
SAVE        access-table-name;
RETRIEVE    access-table-name;
TRANSACTION access-table-name;
REPORT      access-table-name;
FORMAT      access-table-name;

GIVE [ FIRST           ]
     [ LAST            ]
     [ NEXT            ]      access-table-name;
     [ numeric-literal ]
     [ PRIOR           ]
```

```
PUT   [ FIRST          ]
      [ LAST           ] access-table-name;
      [ NEXT           ]
      [ numeric-literal]
      [ PRIOR          ]

STORE [ALL] access-table-name;

UNLOCK;
```

Access Tables.

```
[EXCLUDE] data-element-name;

[ DATE   data name; ]

[ TIME   data-name; ]
```

```
KEY { LIST list-name;
    { data-element-name [UPDATE] [ MODE  UNPACKED  ] ; }
                                 [       BINARY    ]
                                 [       CHARACTER ]

SELECT { WHOLE-FILE;                                                          }
       { PROCEDURE procedure-name[param_1, param_2,..., param_n]; }

[ STRUCTURE[FROM]KEY data-element-name; ]
```

BIBLIOGRAPHY

1. Further Normalization of the Data Base Relational Model.
E.F. Codd.

2. A Data Base Sublanguage Founded on the Relational Calculus.
E.F. Codd.

3. Critique of the Guide/Share D.B.M.S. Requirements.
Gordon C. Everest & Edgar H. Sibley.

4. An Analysis of the April 1971 Data Base Task Group Report.
Robert W. Engles.

5. Computer Data Base Organisation.
James Martin.

6. Computer Data Management and Data Base Technology.
H. Katzan.

7. Principles of Data Base Management.
James Martin.

8. Data Base Management.
J.W. Klimbie and K.L. Koffeman.

9. Human Engineering of Data Base Systems.
S.J. Peck and P.L. Peck.

10. Data Base Management Systems: User Experience in the U.S.A.
B. Davis.

11. Data Base Management.
I. Palmer.

12. File Definition and Logical Data Independence.
C.J. Date & P. Hopewell.

13. Elements of Data Management Systems.
George C. Dodd.

14. Data Base Standardization and Documentation.
Paul L. Peck.

15. The Case for Dedicated Computers.
Richard C. Turnblade.

16. The Integrated Data Base: Need through Implementation.
Paul L. Peck.

17. Auerbach Guide to Data Base Management.

18. The Functional and Organizational Role of the Data Base Administrator.
Paul L. Peck.

19. A Dictionary/Directory Method for Building a Common M.I.S. Data Base.
Don J. Cahill.

20. Fundamentals of D.B.M.S.
Joseph H. Daniel.

21. Information Systems in Perspective.
J.D. Aron.

22. The Use of Distributed Data Bases in Information Networks.
Grayce M. Booth.

23. File Structures for On-line Systems.
D. Lefkovitz.

24. Data Management for On-line Systems.
D. Lefkovitz.

25. Storage Structure and Physical Data Independence.
C.J. Date & P. Hopewell.

26. Data Base Management Systems.
D.A. Jardine.

27. Proceeding of the Data Base Administration Working Group.
Bernard Plagman.

28. Evaluation of Access Authorization Characteristics of Derived Data Sets.
Richard C. Owens Jr.

29. Data Base Management System Evaluation and Selection.
Paul L. Peck.

30. On Interaction with Data Bases.
Naftaly Minsky.

31. A Model for Access Control.
    Peter S. Browne and Dennis D. Steinauer.

32. Using a Structured English Query Language as a Data Definition Facility.
    Raymond F. Boyce & Donald D. Chamberlin

33. The Data Base Task Group Report: An Illustrative Example.
    Randell L. Frank & Edgar H. Sibley.

34. SEQUEL: A Structured English Query Language.

35. Integrity.
    Paul L. Peck.

36. Adaptability to Change in Large Data Base Information Retrieval Systems.
    C.J. Bell, B.K. Aldred, T.W. Rogers.

37. PRTV: A Technical Overview.
    S.J.P. Todd.

38. A Relational Model of Data for Large Shared Data Banks.
    E.F. Codd.

39. Database Sharing: A Study of Interference, Roadblock and Deadlock.
    J.E. Shemer and A.J. Collmeyer.

40. Relational Completeness of Data Base Sublanguages.
    E.F. Codd.

41. Seven Steps to Rendezvous with the Casual User.
E.F. Codd.

42. Normalised Data Base Structure: A Brief Tutorial.
E.F. Codd.

43. Interactive Support for Non-programmers: The Relational and Network Approaches.
E.F. Codd and C.J. Date.

44. Definition of Guidelines for Performing the Six Stages of Data Base System Testing.
Paul L. Peck.

45. Implications of Data Independence on the Architecture of Data Base Management Systems.
Arthur J. Collmeyer.

46. On the Architecture of Data Base Systems.
Domenico Ferrari.

47. Data Base: Techniques and Practices.
British Computer Society.

48. Implementation of Codasyl Data Base Management Proposals.
British Computer Society.

49. Codasyl April 1971 Report.

50. B.C.S. October 1971 Conference on the April 1971 Report.

51. Data Base Management Systems.
Infotech State of the Art Report.

52. On-line Data Bases.
Infotech State of the Art Report.